Architecture des ordinateurs

Le système d'exploitation

Architecture des ordinateurs : le système d'exploitation

Sommaire

Les rôles

Le système d'exploitation (SE) ou OS (***O****perating* ***S****ystem*) est chargé de faire le lien entre :

- Les ressources matérielles (hardware)
- L'utilisateur
- Les applications (software)

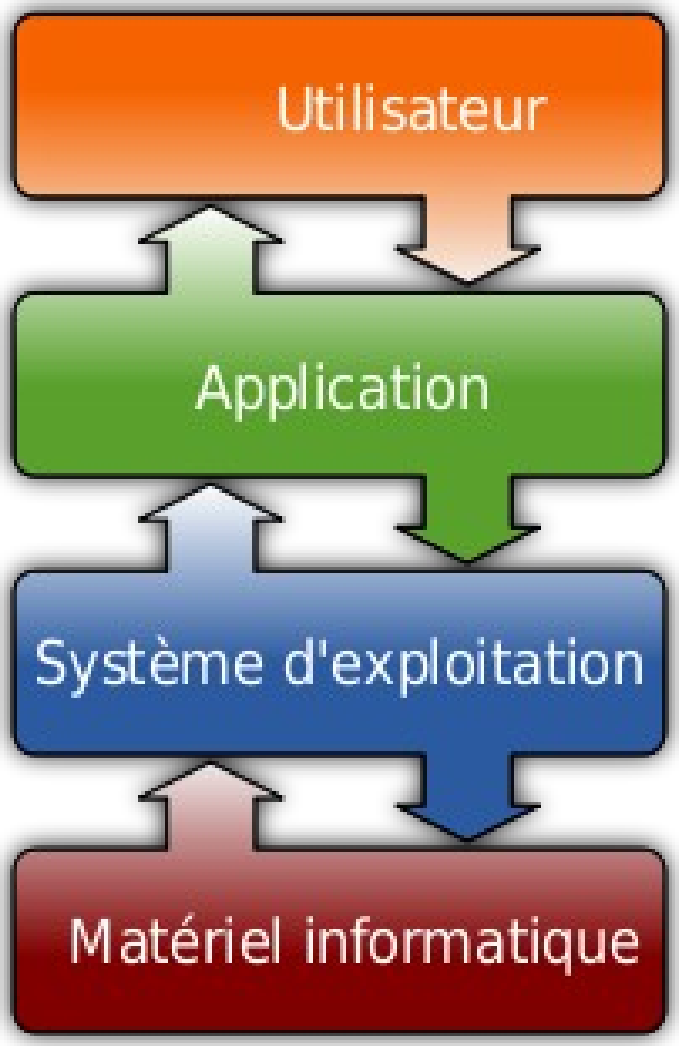
Quand un programme veut accéder au matériel, il envoie les informations au SE qui ''traduit'' au périphérique concerné.

La traduction se fait grâce à du code que l'on appelle pilote ou driver.

Le pilote est développé par le fabricant du matériel.

Architecture des ordinateurs : le matèriel

Les rôles

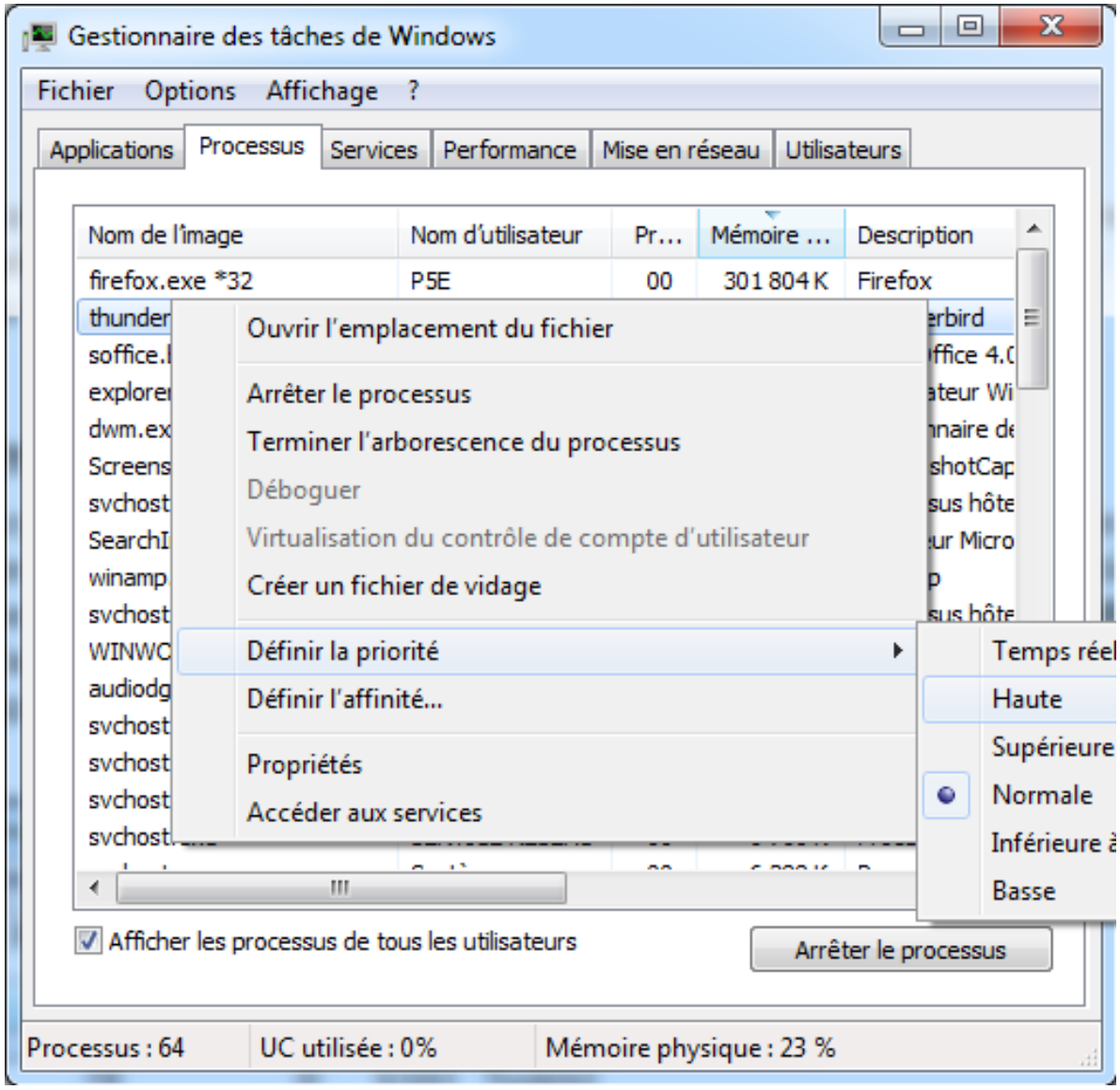


Gestion du processeur :

- Le système d'exploitation est chargé de gérer l'allocation du processeur entre les différents programmes.
- Cette allocation se fait grâce à un **algorithme d'ordonnancement**.
- Le type d'ordonnanceur est totalement dépendant du système d'exploitation.

Architecture des ordinateurs : le matèriel

Les rôles



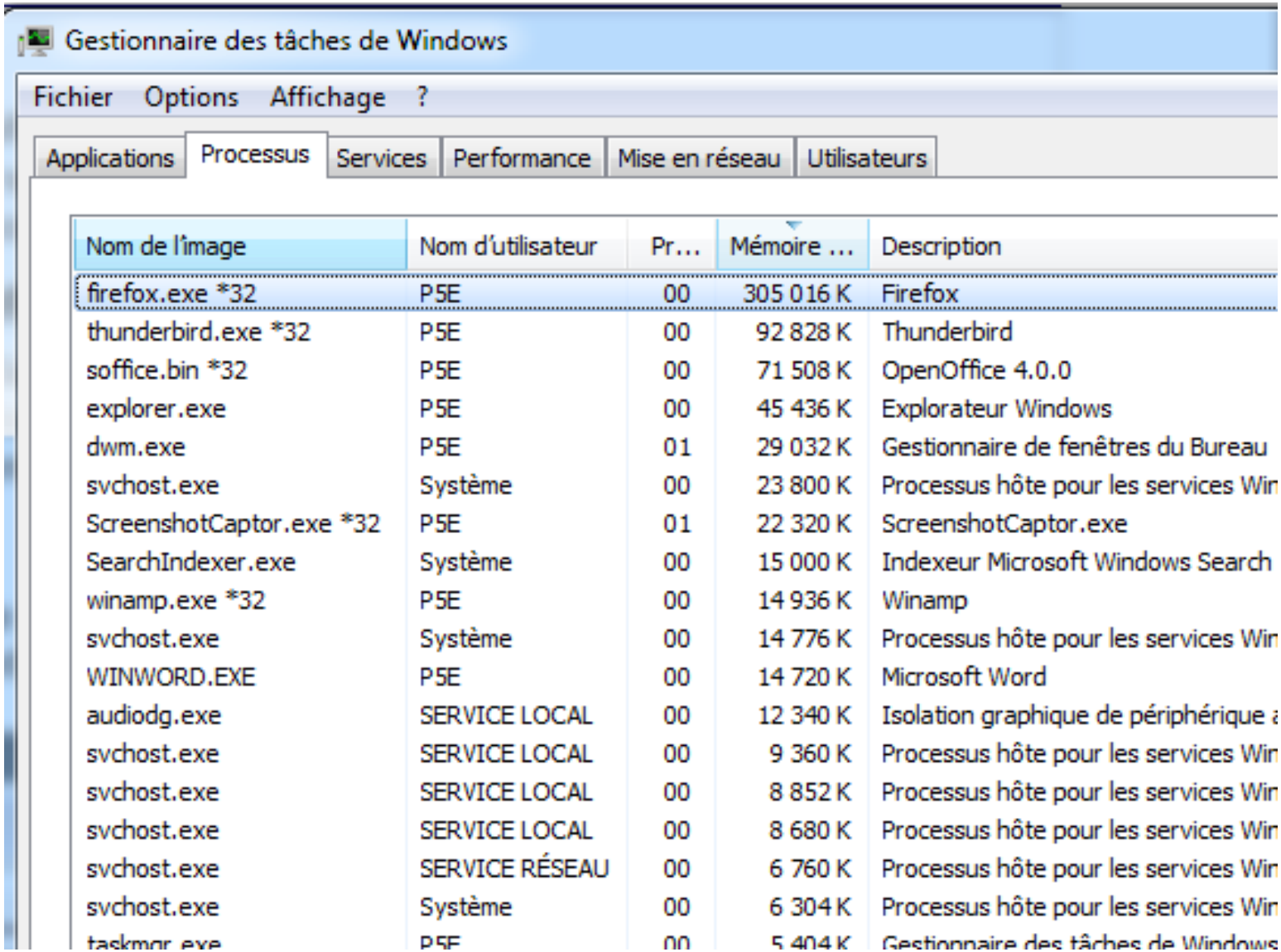
Gestion de la mémoire :

- Le système d'exploitation est chargé de gérer l'espace mémoire alloué à chaque application.

- En cas d'insuffisance de mémoire physique, le système d'exploitation crée une zone mémoire sur le disque dur, appelée «**mémoire virtuelle**».

Architecture des ordinateurs : le matèriel

Les rôles

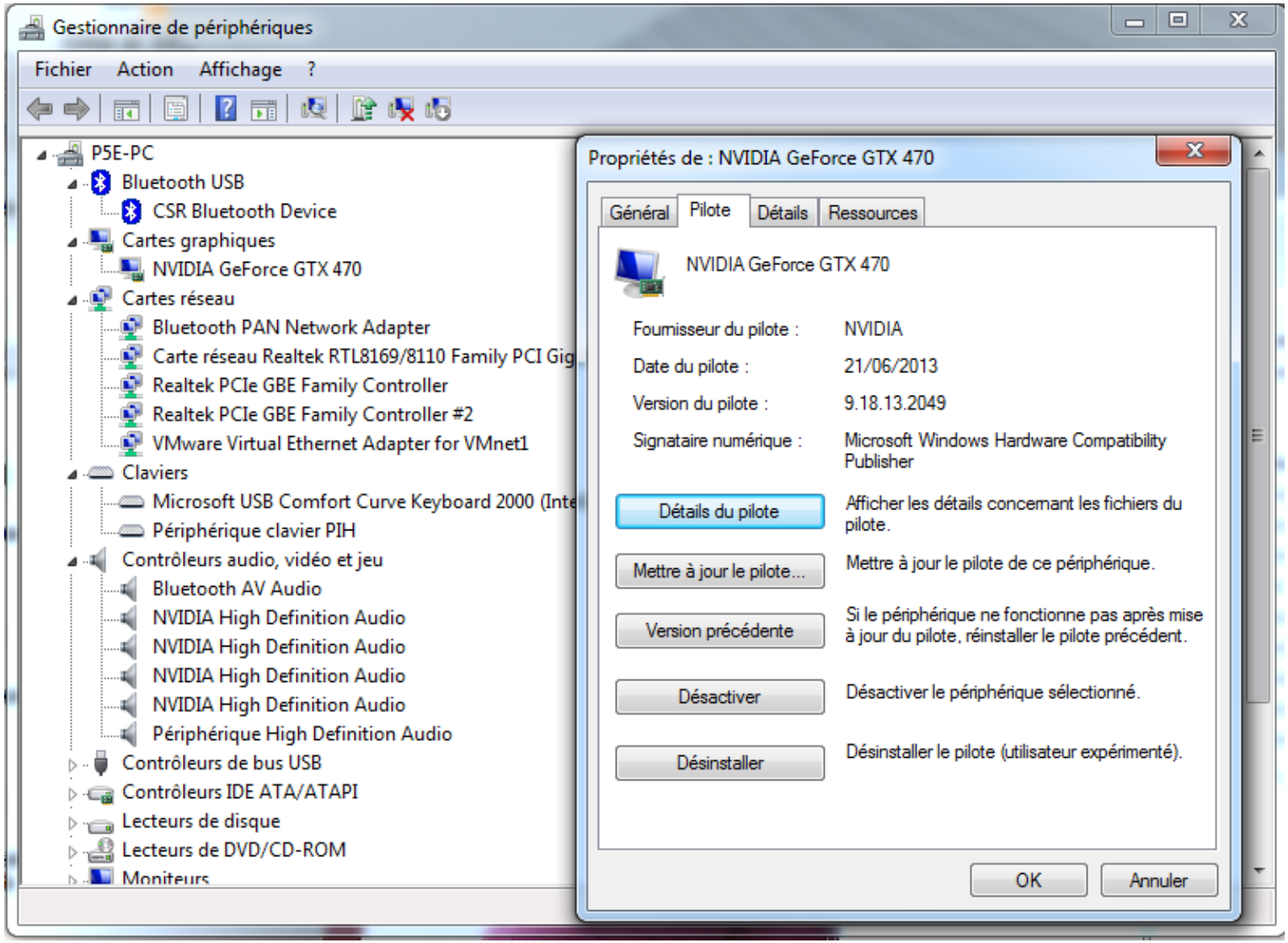


Gestion des entrées/sorties la mémoire :

- Le système d'exploitation permet d'unifier et de contrôler l'accès des programmes aux ressources matérielles.
- Cela se fait par l'intermédiaire des pilotes.

Architecture des ordinateurs : le matèriel

Les rôles

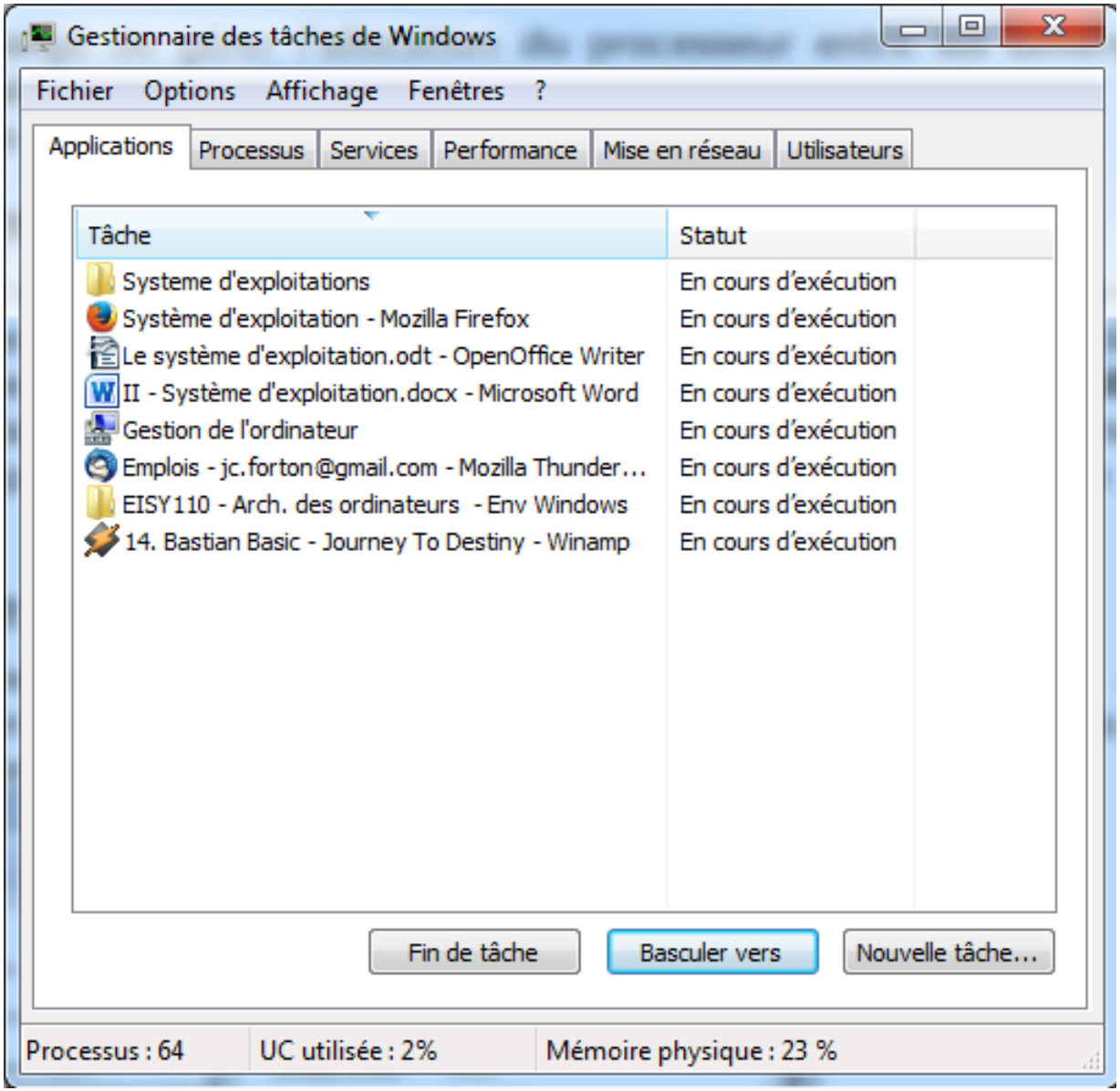


Gestion des applications :

- Le système d'exploitation est chargé de la bonne exécution des applications.
- Il fait cela en leur affectant les ressources nécessaires à leur bon fonctionnement.
- Il permet à ce titre de «tuer» une application ne répondant plus correctement.

Architecture des ordinateurs : le matèriel

Les rôles

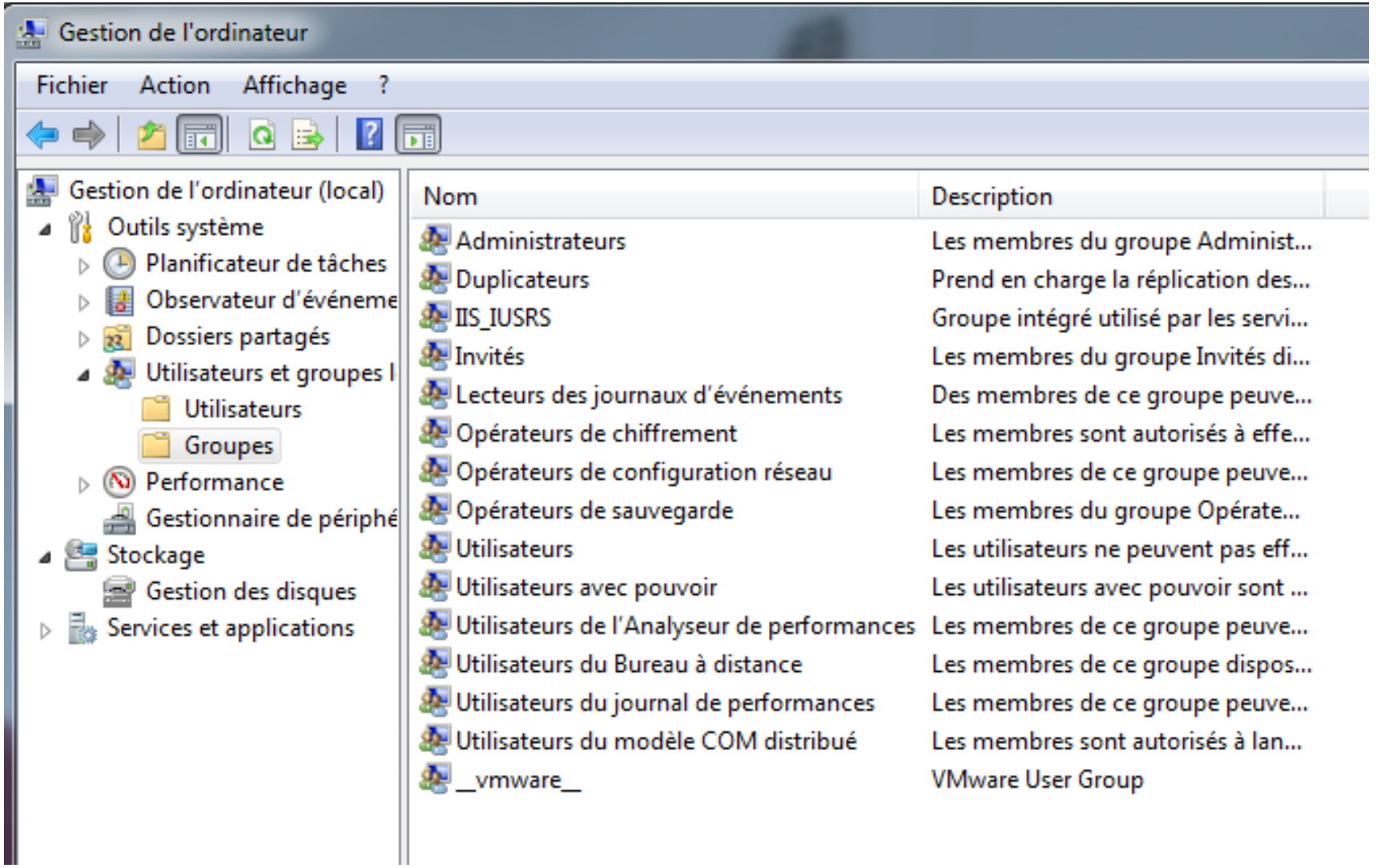


Gestion des droits :

- Le système d'exploitation est chargé de la sécurité liée à l'exécution des programmes.
- Il fait cela en garantissant que les ressources ne sont utilisées que par les programmes et utilisateurs possédant les droits adéquats.

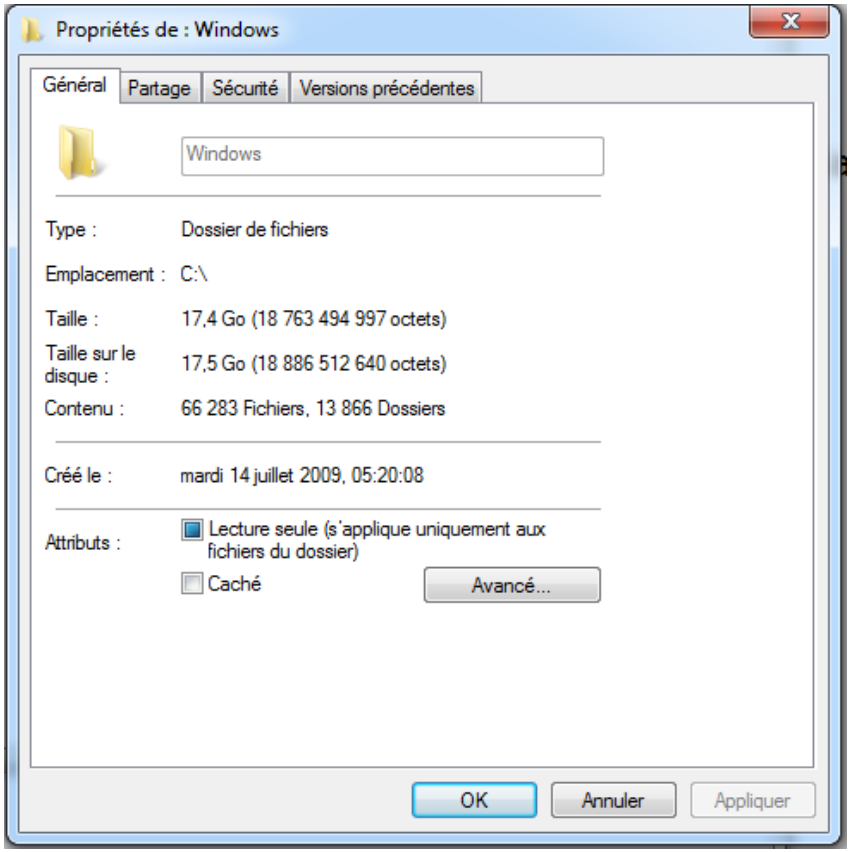
Architecture des ordinateurs : le matèriel

Les rôles



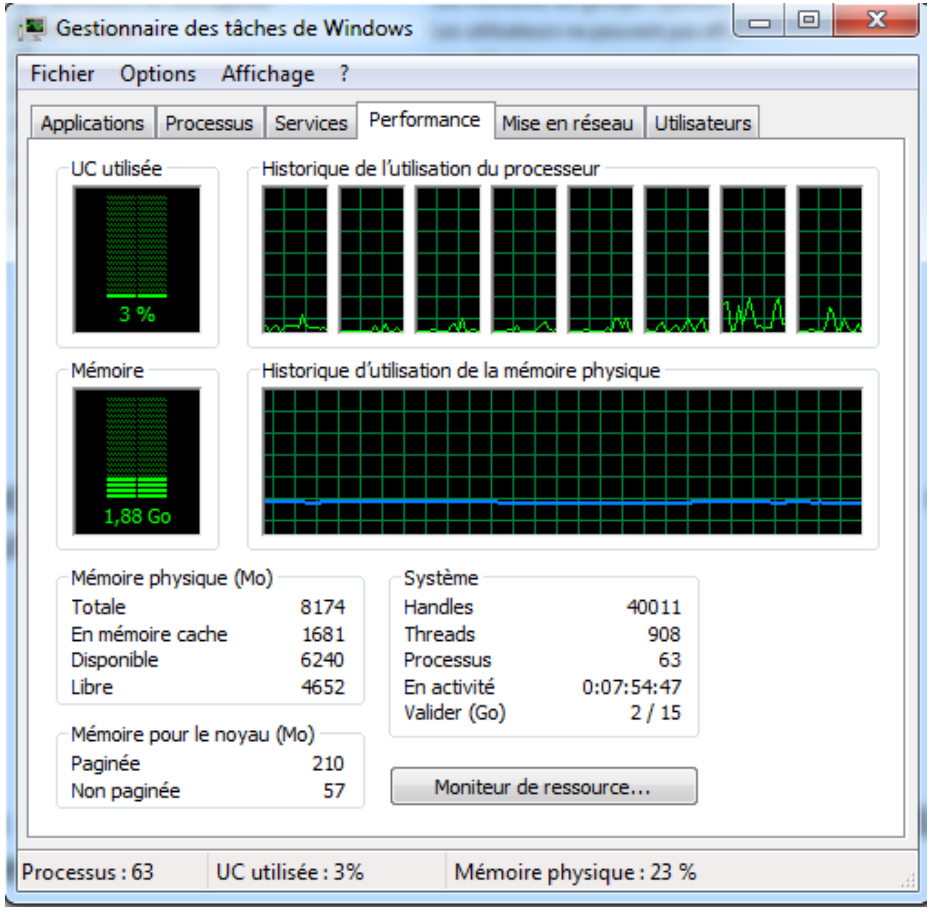
Gestion des fichiers :

- Le système d'exploitation gère la lecture et l'écriture dans le système de fichiers et les droits d'accès aux fichiers par les utilisateurs et les applications.



Gestion des informations :

- Le système d'exploitation fournit des indicateurs permettant de diagnostiquer le bon fonctionnement de la machine.



Les différents aspects des systèmes

L'aspect multitâche

- Le ***multitasking*** est la capacité d'exécuter, de façon apparemment simultanée, plusieurs programmes informatiques.
- Cette fonction est **indépendante du nombre de processeurs** présents physiquement dans l'ordinateur.
- La simultanéité apparente est le résultat de l'alternance rapide d'exécution des processus présents en mémoire.

L'aspect multitâche

- Le passage de l'exécution d'un processus à un autre est appelé **commutation de contexte**.

- Ces commutations peuvent être initiées par les programmes eux-mêmes (***multitâche coopératif***) ou par le système d'exploitation lors d'événements externes (***multitâche préemptif***).

- Le multitâche préemptif est plus robuste que le multitâche coopératif car une tâche ne peut bloquer l'ensemble du système.

L'aspect multi-utilisateur

- **Plusieurs utilisateurs peuvent utiliser l'ordinateur simultanément.**
- **Implique de limiter les droits d'accès de chacun afin de garantir l'intégrité des données.**
- **Opposé à mono-utilisateur (utilisable par un seul utilisateur)**

L'aspect physique

- Système centralisé :

Le système ne gère que les ressources de la machine sur laquelle il est présent.

- Système réparti :

Ensemble de machines autonomes connectées par un réseau, et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources.

L'aspect architecture matérielle

- Système multiprocesseurs (***S***ymetric **M**ulti**p**rocessor)
- *Systèmes temps réels*

L'aspect architecture matérielle

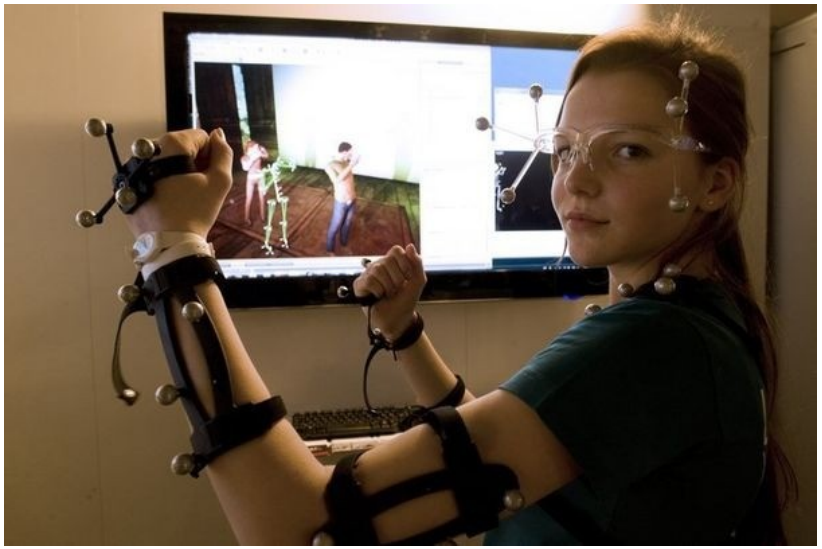
- Système multiprocesseurs (SMP) :
  - → permet un parallélisme de tâches
  - → un processus peut être exécuté sur chaque processeur
  - → plus grande puissance de calcul qu'en monoprocesseur

Soit plusieurs programmes disposent d'un processeur

Soit le développeur conçoit son programme pour tirer partie de cette puissance de calcul

L'aspect architecture matérielle

- Système temps réel :
  - Essentiellement utilisé dans l'industrie
  - Fonctionne dans un environnement contraint temporellement
  - Nécessite un SE spécial : RTOS (**R**eal-**T**ime **O**perating **S**ystem)



La gestion des ressources matérielles

La principale fonction d'un système d’exploitation est de lancer les programmes et **répartir les ressources** :

- processeur
- mémoire
- périphériques
- …

entre les différents programmes qui s’exécutent en même temps, et cela **de manière efficace et harmonieuse**.

Il doit gérer les **entrées-sorties**

- prendre en charge le transfert d'information entre :
  - l'unité centrale
  - les périphériques
  - le réseau

- Configurer le matériel par le biais de fichiers systèmes

- Gérer les échanges entre ces composants

Il doit gérer les **interruptions matérielles**

- Une IRQ (**I**nterrupt **R**e**q**uest) est un signal en provenance d'un périphérique à destination du SE.
- Le SE intercepte les *IRQ.*
- Les *IRQ* sont utilisées en informatique pour réagir en temps réel à un événement asynchrone.
- Cela permet aussi d'économiser le temps d'exécution lié à une boucle de consultation (polling loop).

```
function action(){
    ...
}

attachInterrupt(touche du clavier appuyée, action)
```

```
function action(){
    ...
}

while(true){
    if(touche du clavier appuyée){
        action()
    }
}
```

Il doit gérer **les processus.**

- **Un processus est un programme qui est en cours d'exécution.**
- Son exécution dure un certain temps avec un début et (parfois) une fin.
- Les applications utilisateur sont des (ensembles de) processus.

explorer.exe	Deejc	00	52 260 K	Explorateur Windows
firefox.exe	Deejc	00	80 668 K	Firefox
firefox.exe	Deejc	00	89 764 K	Firefox
firefox.exe	Deejc	01	29 300 K	Firefox
firefox.exe	Deejc	00	130 324 K	Firefox
firefox.exe	Deejc	00	223 484 K	Firefox
firefox.exe	Deejc	00	130 672 K	Firefox
firefox.exe	Deejc	00	215 060 K	Firefox

- Un processus doit être chargé en mémoire centrale pour pouvoir s'exécuter (dans la RAM).

Il peut gérer **le multitâches.**

- Plusieurs programmes s'exécutent en même temps, cela nécessite de :
  - → Gérer l'allocation mémoire
  - → Partager le processeur entre les processus

- Un système préemptif utilise un ordonnanceur qui répartit le temps machine entre les différents processus selon des critères de priorité.

- Un système coopératif laisse les processus se répartir le temps entre eux.

La gestion des fichiers

Le système d'exploitation sert d'intermédiaire entre le haut niveau (les applications) et le bas niveau (les pilotes des disques).

La partie du système d’exploitation qui se charge de cela se nomme **système de gestion de fichiers** (***SGF***).

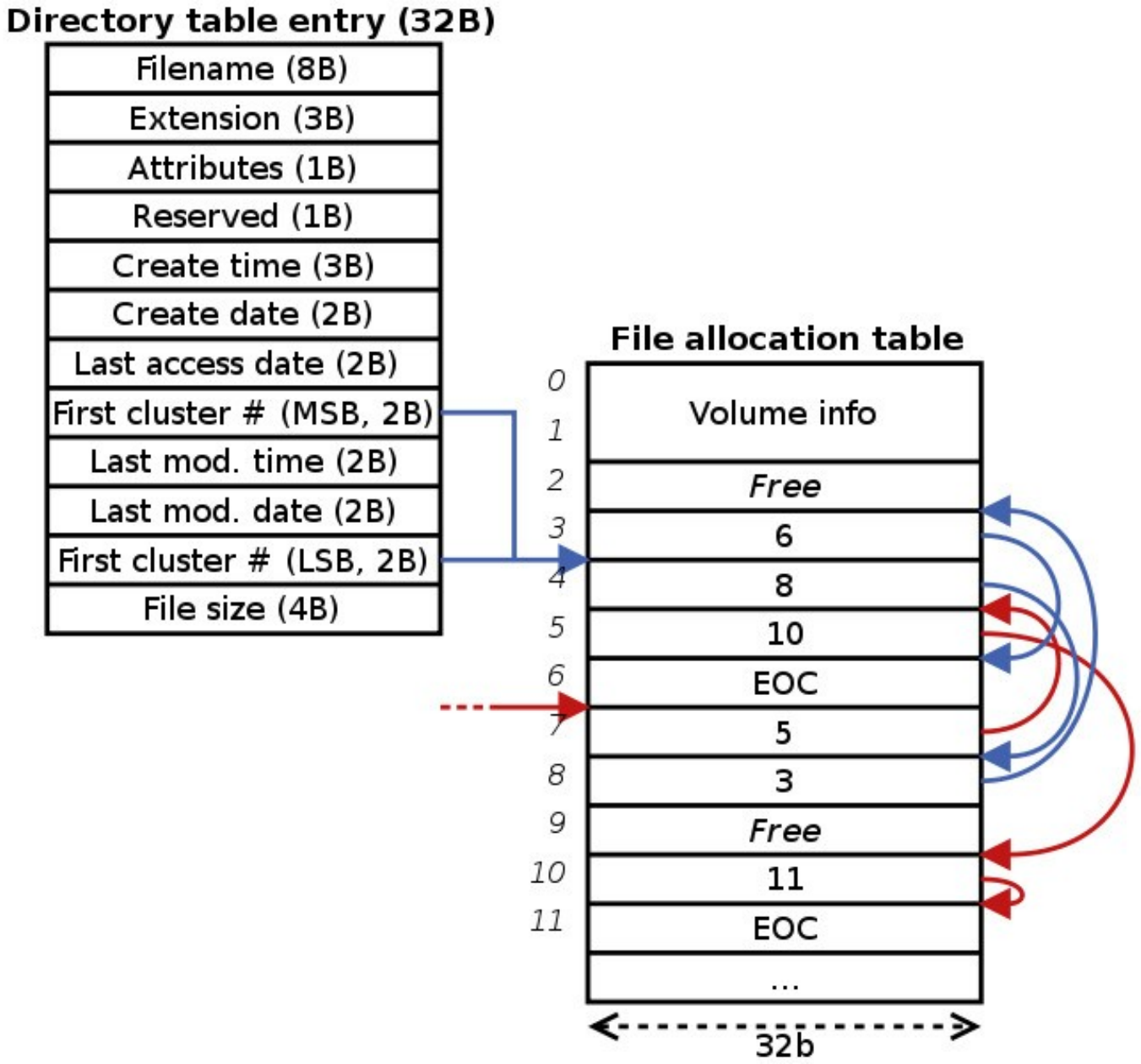
Un *SGF* est une façon de stocker les informations et de les organiser.

Le SGF permet de conserver les données ainsi que de les partager entre plusieurs programmes informatiques.

Le SGF offre à l'utilisateur une vue **abstraite** sur ses données et permet de les localiser à partir d'un chemin d'accès.

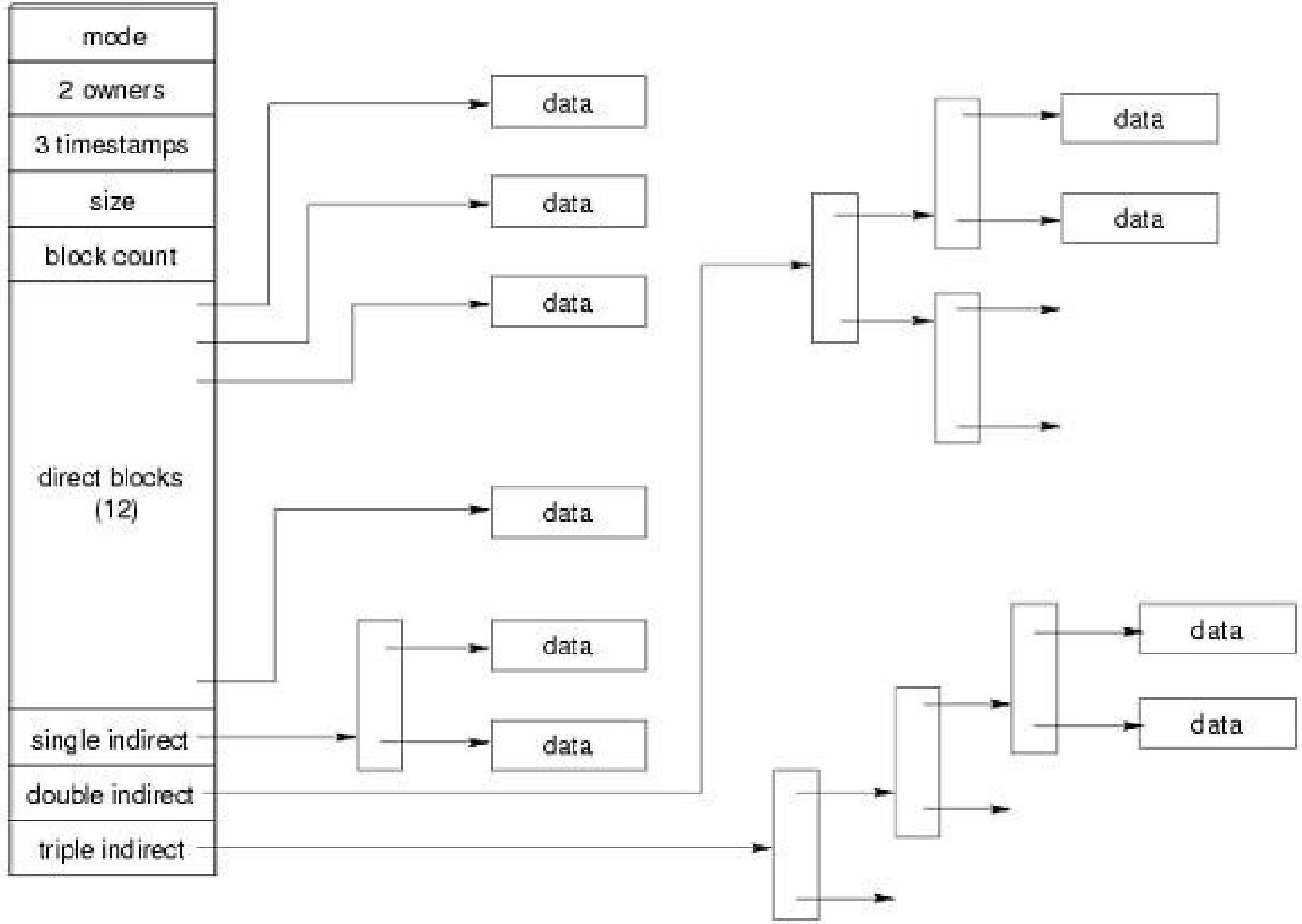
Sous Windows (File Allocation Table) :

- Chaque répertoire contient une table associant les noms de fichiers à leur taille.
- Cette table contient un index pointant vers la table d'allocation de fichiers.
- La table d'allocation est une zone réservée du disque.
- La table d'allocation indique pour chaque bloc de données l'index du bloc suivant du même fichier.


Directory table entry (32B)
Filename (8B)
Extension (3B)
Attributes (1B)
Reserved (1B)
Create time (3B)
Create date (2B)
Last access date (2B)
First cluster # (MSB, 2B)
Last mod. time (2B)
Last mod. date (2B)
First cluster # (LSB, 2B)
File size (4B)
File allocation table
0
1
2
3
4
5
6
7
8
9
10
11
Volume info
Free
6
8
10
EOC
5
3
Free
11
EOC
...
32b

Sous Linux (Extended FileSystem) :

- les fichiers et les répertoires sont identifiés par un numéro unique, le numéro **d'inode.**
- Ce numéro permet d'accéder à une structure de données (**inode**) regroupant toutes les informations :
  - la protection d'accès en lecture / écriture
  - des listes de dates de création, modification
  - Un moyen de retrouver le contenu
  - …
- Le nom du fichier est stocké dans le répertoire associé à un numéro d'inode.
- L'avantage est qu'un fichier unique sur le disque peut être connu du système sous plusieurs noms (liens symboliques).


mode
2 owners
3 timestamps
size
block count
direct blocks
(12)
single indirect
double indirect
triple indirect
data
data
data
data
data
data
data
data
data
data

La fragmentation, principe :

- Le disque dur est divisé en **secteurs**.
- Le système d'exploitation les rassemble pour former des **clusters** (blocs).
- Chaque fichier utilise plusieurs clusters.
- Si les **clusters** contenant le fichier sont **contigus**, celui-ci n'est **pas fragmenté**.
- Les clusters d'un fichier sont généralement **éparpillés par groupes**, d'où la **fragmentation** du fichier.

La fragmentation, cas de Windows :

- Le noyau NT essaie de combler les blocs vides.
- Il fragmente le fichier pour ne pas laisser au début du disque des zones avec des clusters libres.
- La fragmentation survient après de multiples suppressions, modifications, copies de fichiers sur le disque dur.
- Cela favorise l'apparition de zones de clusters libres, et par conséquent la fragmentation.

La fragmentation, cas de Linux :

- Le noyau calcule le nombre de blocs nécessaires au stockage de chaque fichier.
- Si le nombre de clusters libres contigus est trouvé sur le disque, il stocke le fichier sur ces clusters et le fichier n'est pas fragmenté.
- S'il ne trouve pas assez de clusters libres contigus, il scinde le fichier en plusieurs groupe de clusters.
- Il tente de minimiser le nombre de groupes.
- Les plus grands espaces de clusters vides contigus sont remplis en premier.

La gestion de la mémoire

La gestion de la mémoire est un compromis entre les performances (temps d'accès) et la quantité (espace disponible) :

- On désire une capacité maximum de mémoire
- On désire y accéder très rapidement

C'est le rôle de la *MMU* (**M**emory **M**anagement **U**nit) :

- partager de la mémoire (pour un système multitâches)
- allouer de la mémoire aux différentes tâches
- protéger les espaces mémoire utilisés
- optimiser la quantité de mémoire disponible

Pour y arriver, la *MMU* utilise deux techniques :

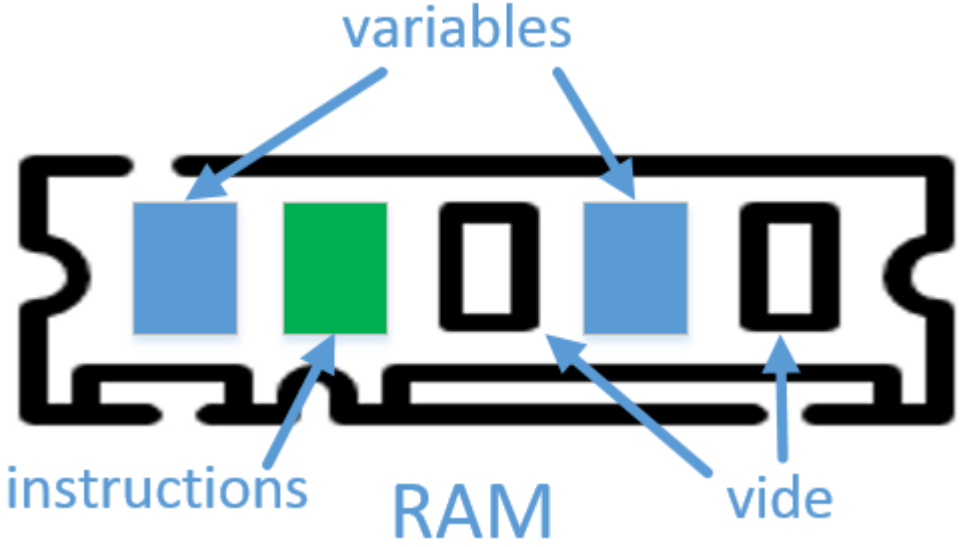
- la segmentation
- la pagination

Gestion avec segmentation

La segmentation permet la séparation des données et du programme dans des espaces logiquement indépendants.

La segmentation permet une plus grande protection grâce au niveau de privilège de chaque segment.

Cela facilite la programmation, l'édition de liens et le partage inter-processus.

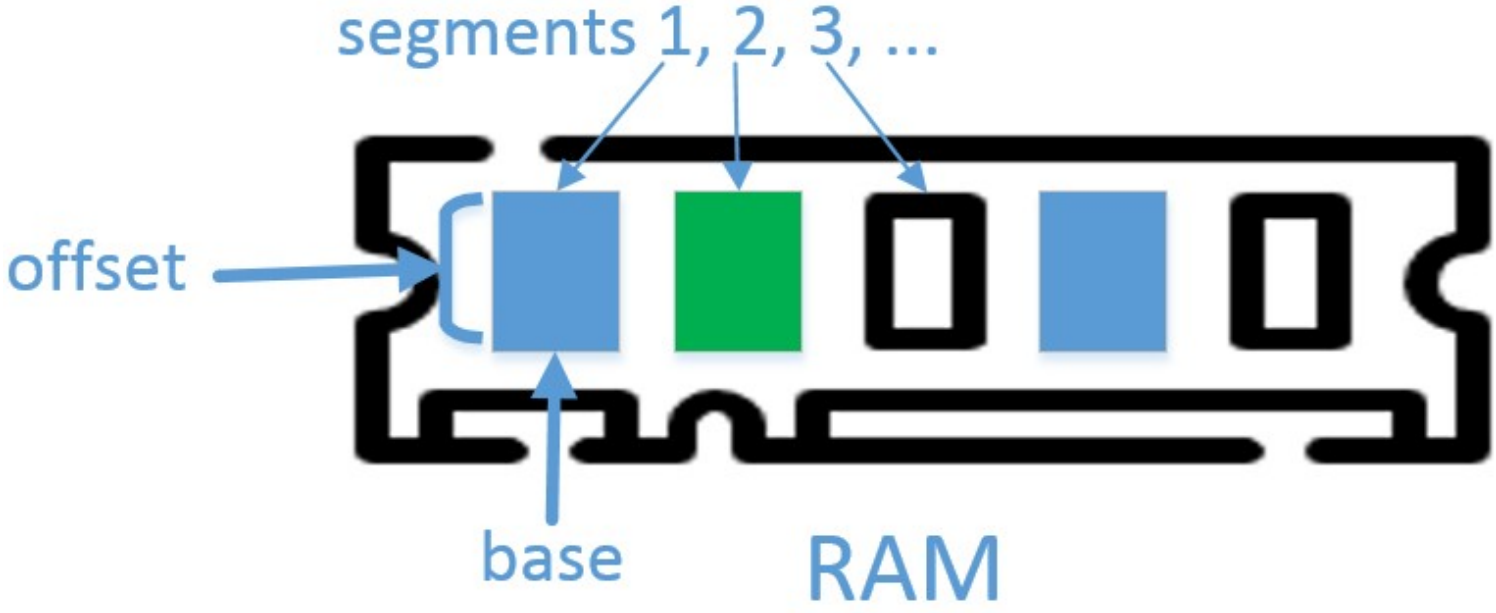


Gestion avec segmentation

Un segment mémoire est défini par deux valeurs :

- L'adresse où il commence (***base***)
- Sa taille ou son *décalage* (*limite* ou ***offset***)

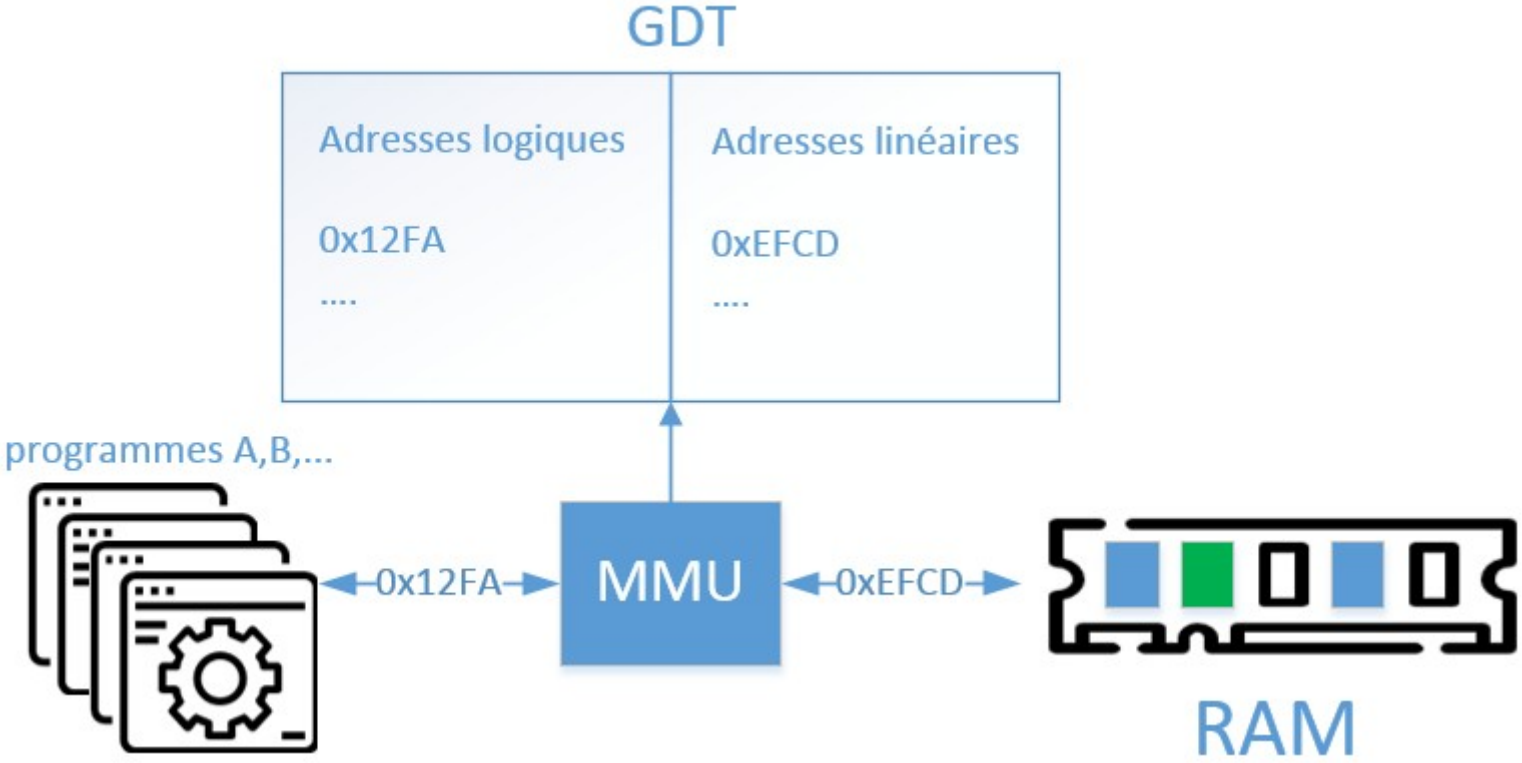
On remarque que les segments sont disposés de manière linéaire, c'est pourquoi on appelle leurs adresses des adresses linéaires.



Gestion avec segmentation

La *MMU* ne va pas communiquer les adresses des segments aux programmes mais des adresses logiques.

La *MMU* stocke la correspondance entre adresses linéaires et logiques dans une table : la *GDT* (***G**lobal **D**escriptor **T**able*).



Gestion avec segmentation

L'avantage pour le programmeur est qu'il n'a pas à connaître la quantité de mémoire dont dispose l'ordinateur sur laquelle son programme va s'exécuter, il demande seulement une quantité de mémoire.

Un autre avantage est que l'on peut modifier la quantité de mémoire physique de l'ordinateur sans que cela impacte le fonctionnement des programmes.

Gestion avec pagination

La gestion avec pagination permet de résoudre la problématique suivante :

Que se passe-t-il si mes programmes demandent plus de mémoire que disponible ?

Dépasser la quantité de mémoire physique de l'ordinateur est possible grâce à la pagination.

Gestion avec pagination

Avec cette technique, on définit :

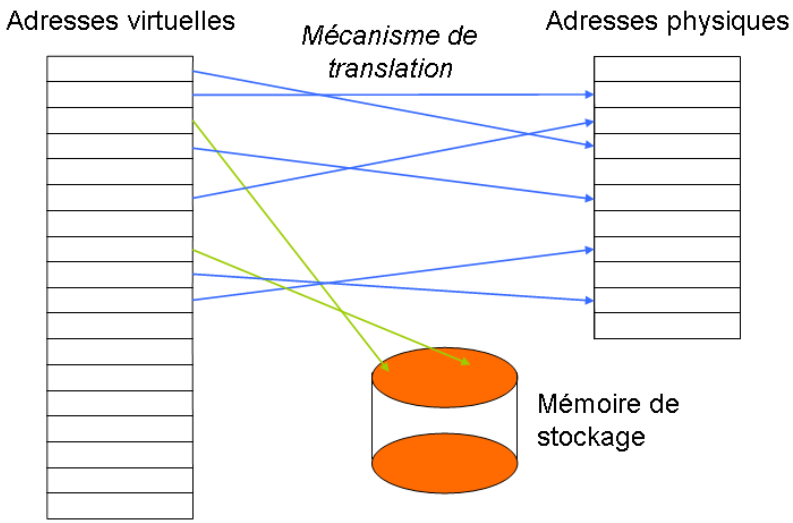
- des pages (mémoire virtuelle)
- des cadres (blocs de mémoire physique de 1, 2 ou 4 ko)

Les pages peuvent être stockées sur le disque dur.

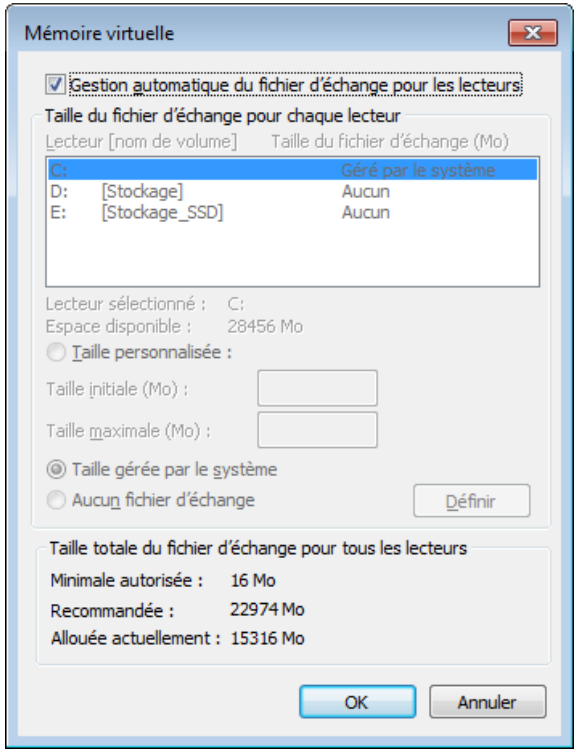
Les pages doivent être chargées dans des cadres pour être utilisées.

La translation entre adresses virtuelles et adresses physiques est gardée dans une table de pages.

Gestion avec pagination



Le stockage des pages sur le disque dur se fait dans un fichier temporaire appelé fichier ***SWAP*** quand la mémoire vive n'est plus suffisante.



Cela induit une **baisse considérable des performances**, étant donné que le temps d'accès du disque dur est extrêmement plus faible que celui de la RAM.

Les composants

Le système d'exploitation permet de gérer les interactions avec le matériel grâce aux composants suivants :

- Le **noyau** (**kernel**) représentant les fonctions fondamentales du système d'exploitation.

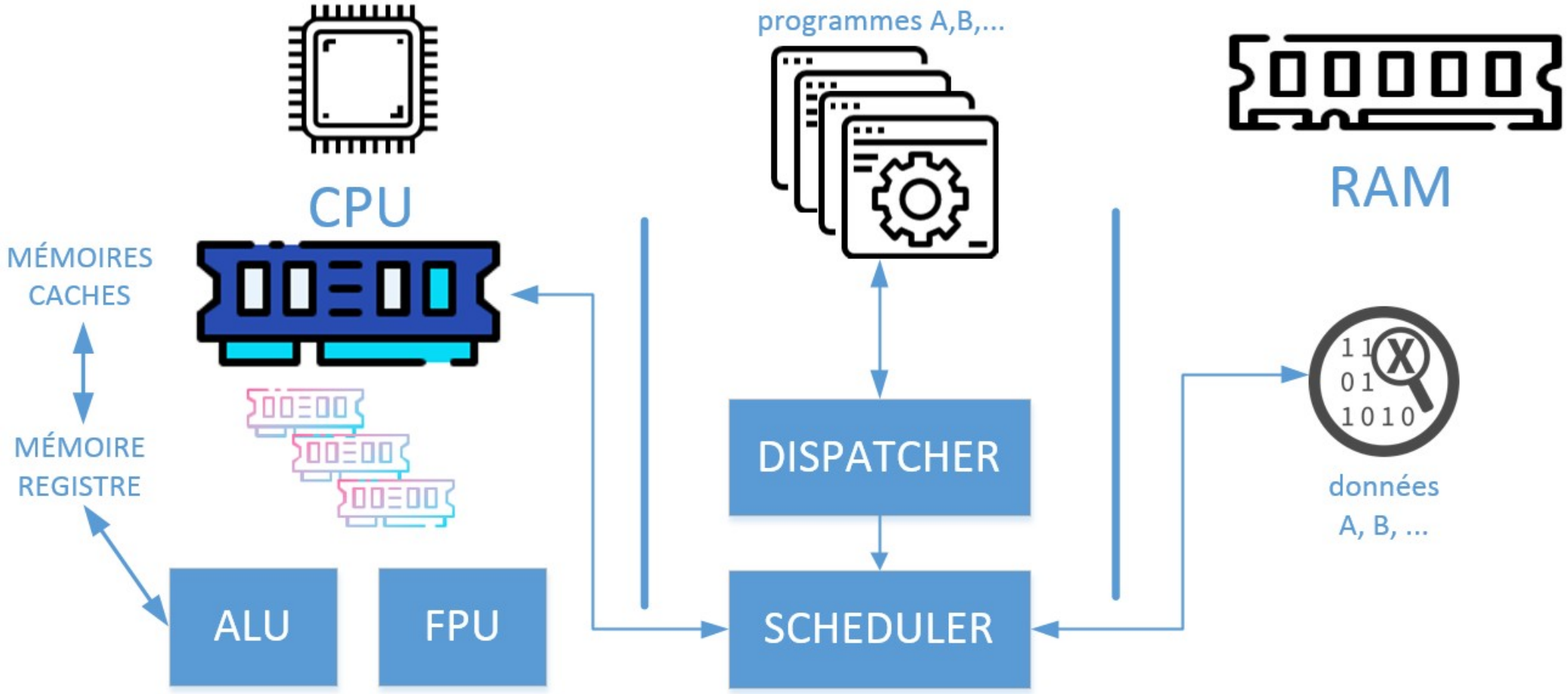
- L'**interpréteur de commande** (**shell**) permettant la communication avec le système d'exploitation par l'intermédiaire d'un langage de commandes.

- Le **système de fichiers** (**file system**, noté FS), permettant d'enregistrer les fichiers dans une arborescence.

Le noyau est composé de plusieurs modules qui assurent chacun une fonctionnalité :

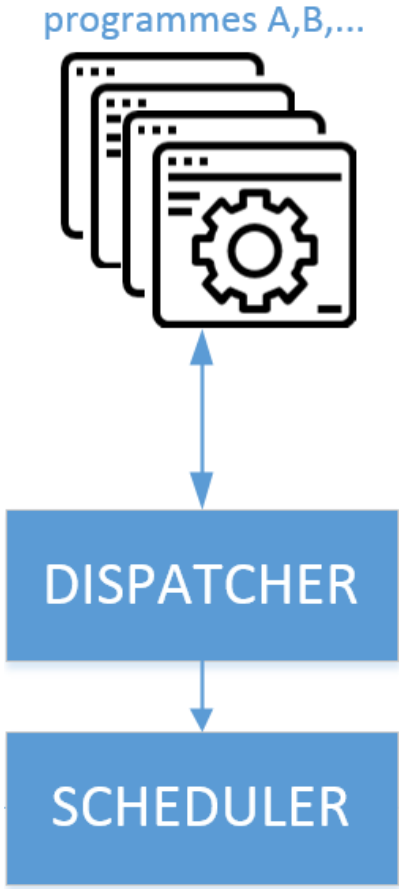
- L'allocateur (***dispatcheur***) :
  - ➔ responsable de la sélection des processus à exécuter
- Le planificateur (**scheduler**) :
  - ➔ répartit le temps disponible de l'unité de traitement entre les différents processus (affecte une priorité)
  - ➔ sauvegarde d'état lorsque le processus s'interrompt
  - ➔ indique au CPU le processus suivant
- Le gestionnaire d'interruptions :
  - ➔ détermine la source de l'interruption
  - ➔ active la procédure associée

Intéressons-nous au couple *dispatcher / scheduler* :



Prenons l'exemple suivant :

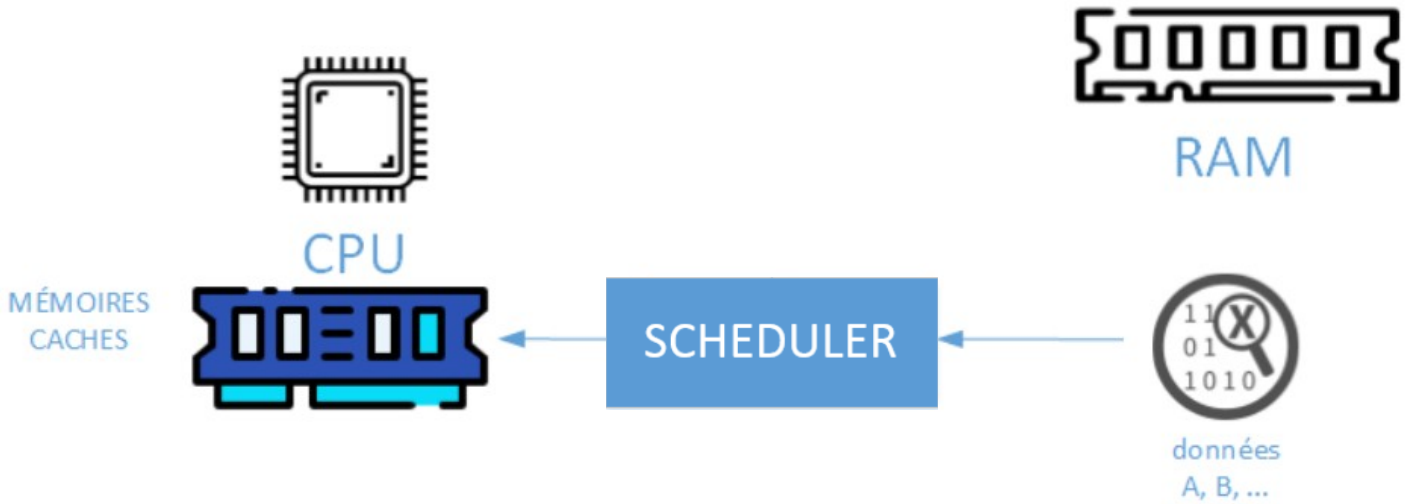
1)Le *dispatcher* décide, dans l'ensemble des programmes en exécution, que c'est au programme B de s'exécuter et l'envoie au *scheduler.*



2) Le programme B donne ses instructions au *scheduler* :
   - add,r1,r2,r3 → r3 = r1+r2
   - cmp,r3,r4,r5 → compare r3 et r4 et met le résultat dans r5
   - ...

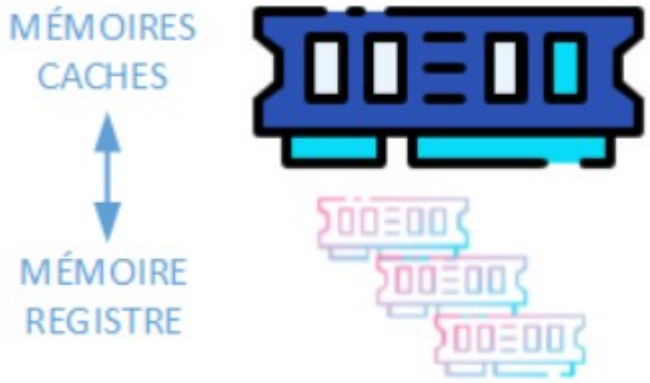
3) Le *scheduler* récupère les adresses de :
   - r1,r2 et r4

4)Le *scheduler* copie les valeurs de r1,r2 et r4 de la RAM vers la mémoire cache

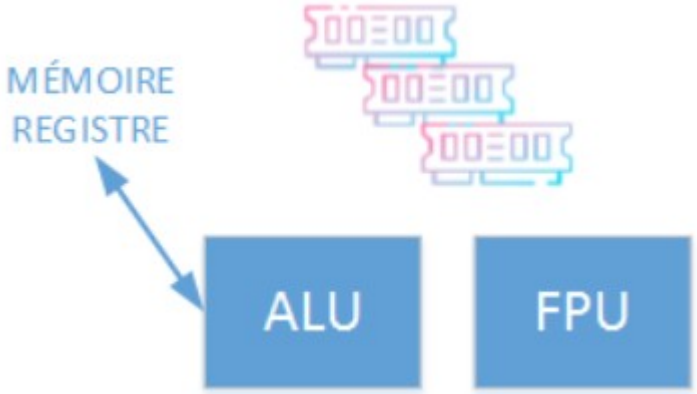


5) Le *scheduler* demande au CPU de traiter les instructions du programme B (en utilisant un registre spécial : le compteur ordinal)

6)Le CPU ''descend'' les variables r1,r2 et r4 du cache vers le registre pour effectuer les traitements demandés

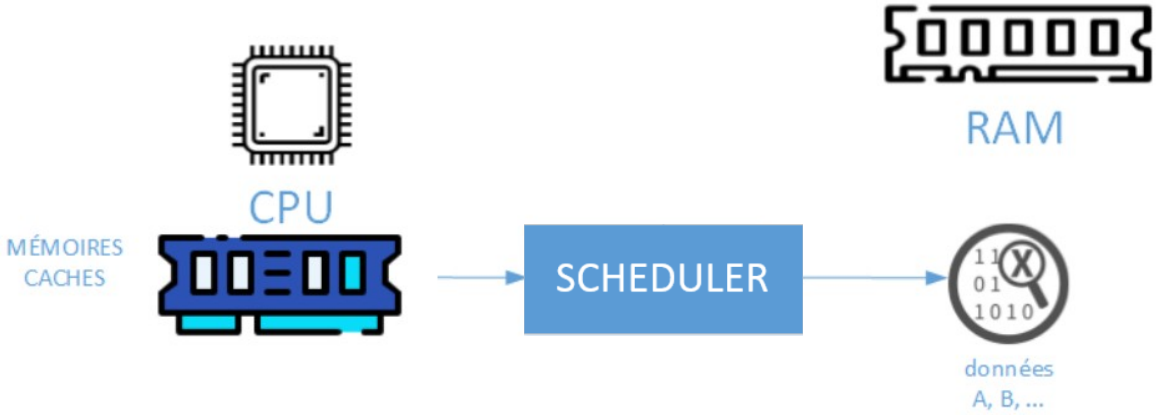


7)L'ALU et la FPU effectuent les opérations demandées et remontent les résultats dans les registres.



8)Le dispatcher décide que c'est au programme Z de s'exécuter et donne l'ordre au CPU de remonter les informations concernant B des registres vers les caches.

6)Le *scheduler* décharge les informations du programme B du CPU pour les remettre dans la RAM (sinon elles sont perdues).



7)Le même processus recommence pour le programme Z !

8)Quand le *scheduler* décharge les données d'un processus pour charger celles d'un autre, cela s 'appelle la **commutation de contexte**.

Intéressons-nous maintenant au gestionnaire d'interruption.

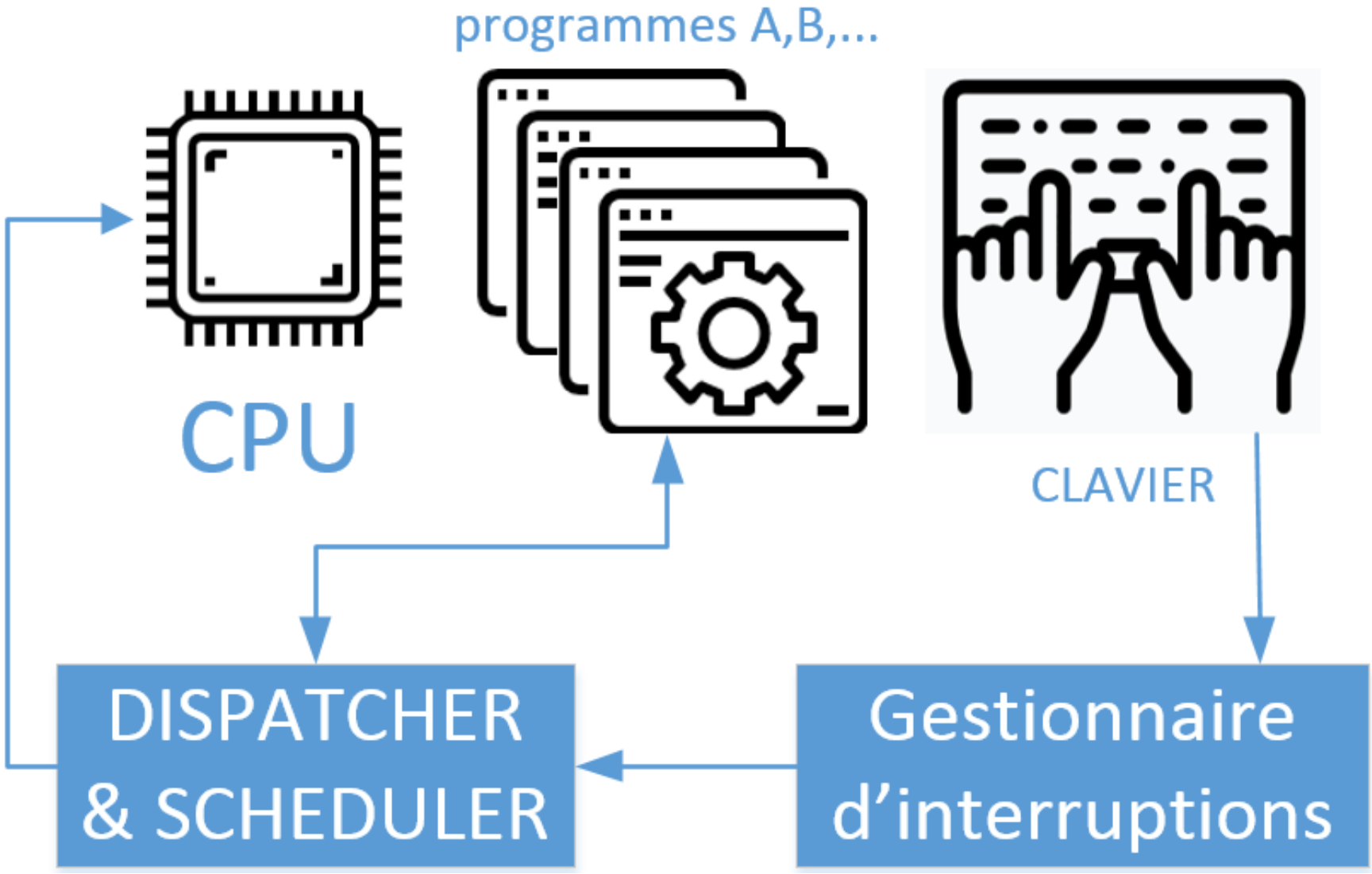
Pour comprendre la notion d'interruption, prenons le scénario suivant :

On utilise un logiciel de traitement de texte tout en écoutant de la musique, surfant sur Internet, etc.

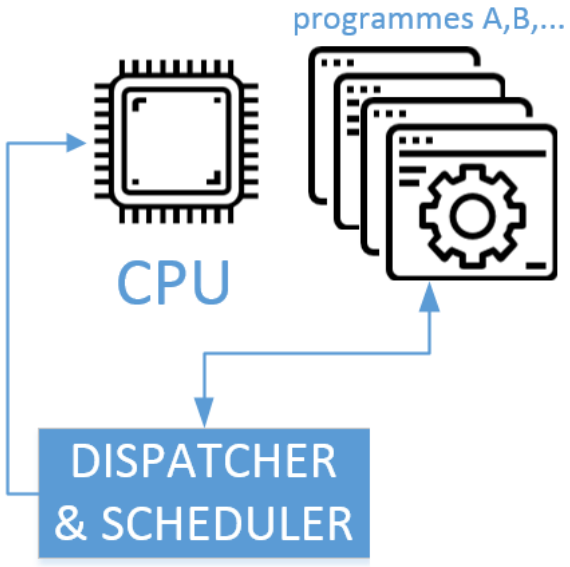
Si on décompose la situation :

- Le logiciel de traitement de texte attend des caractères.
- On utilise le clavier pour envoyer des caractères.
- D'autres programmes monopolisent le CPU pour s'exécuter.

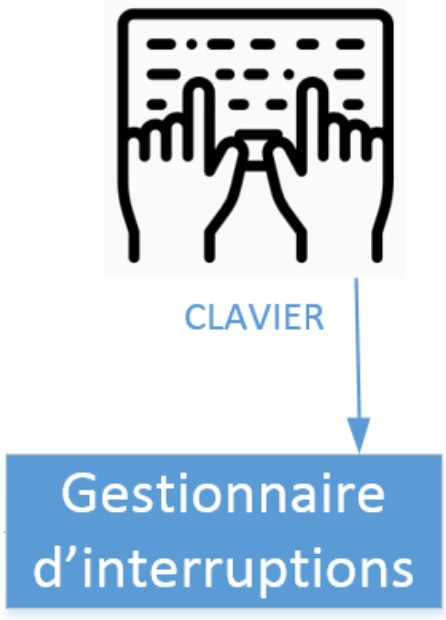
La configuration est la suivante :



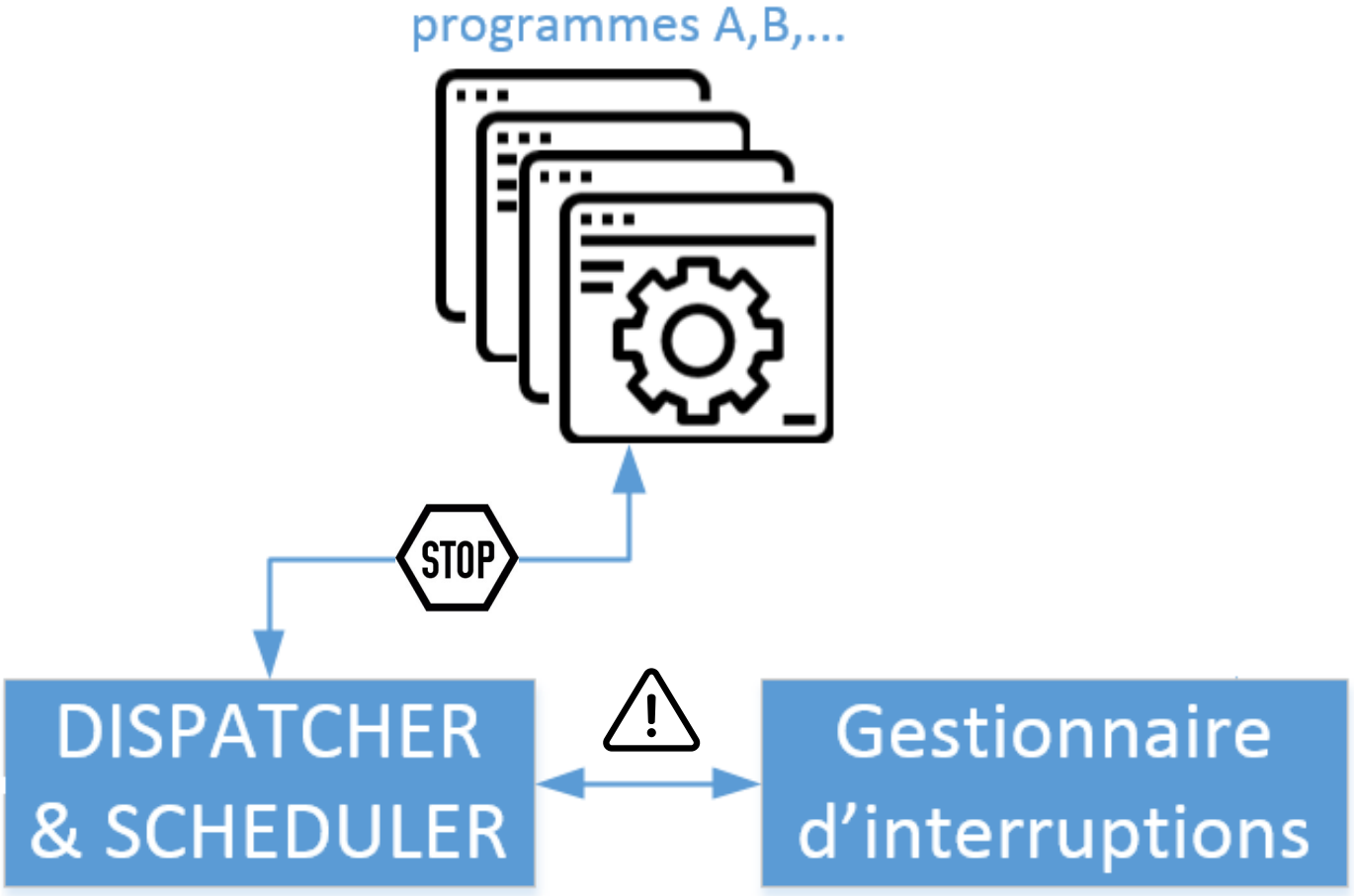
1)Le *dispatcher* avec le *scheduler* sont pris dans leur activité de répartition du temps d'exécution des programmes…



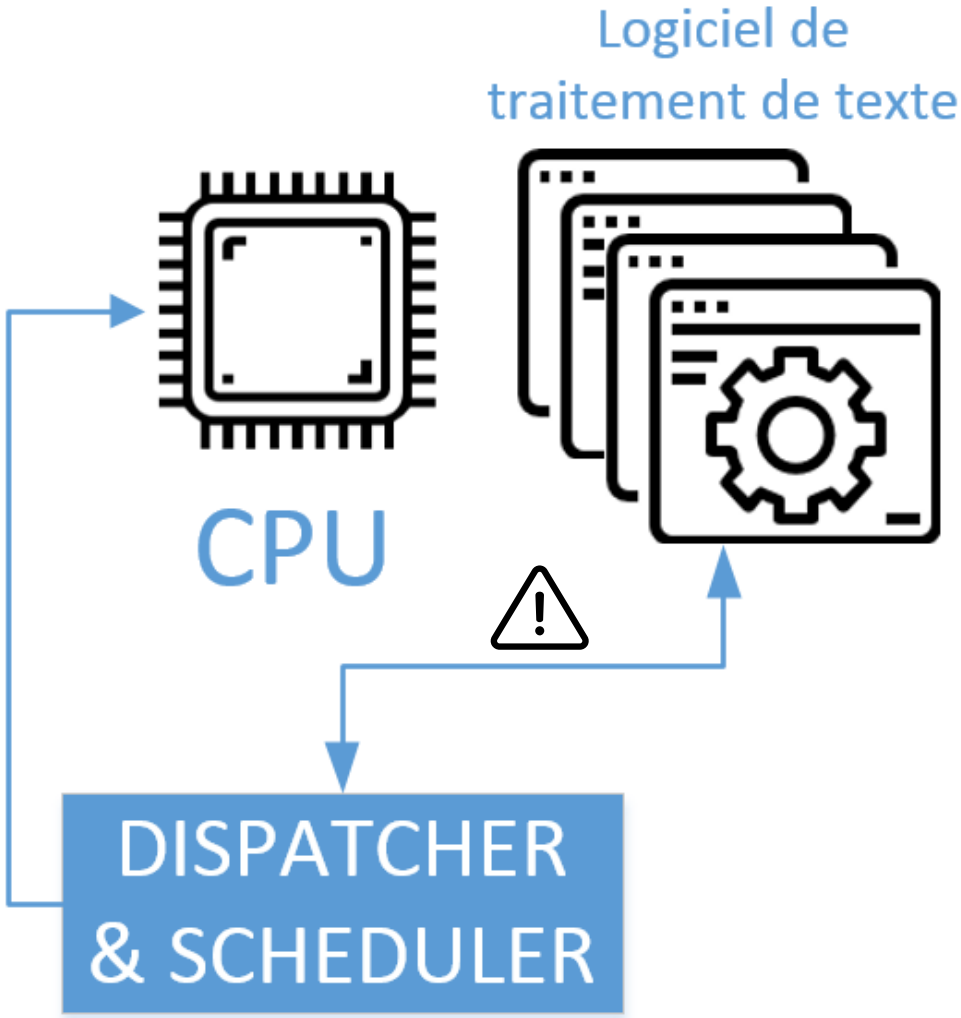
2)L'utilisateur appuie sur une touche du clavier.



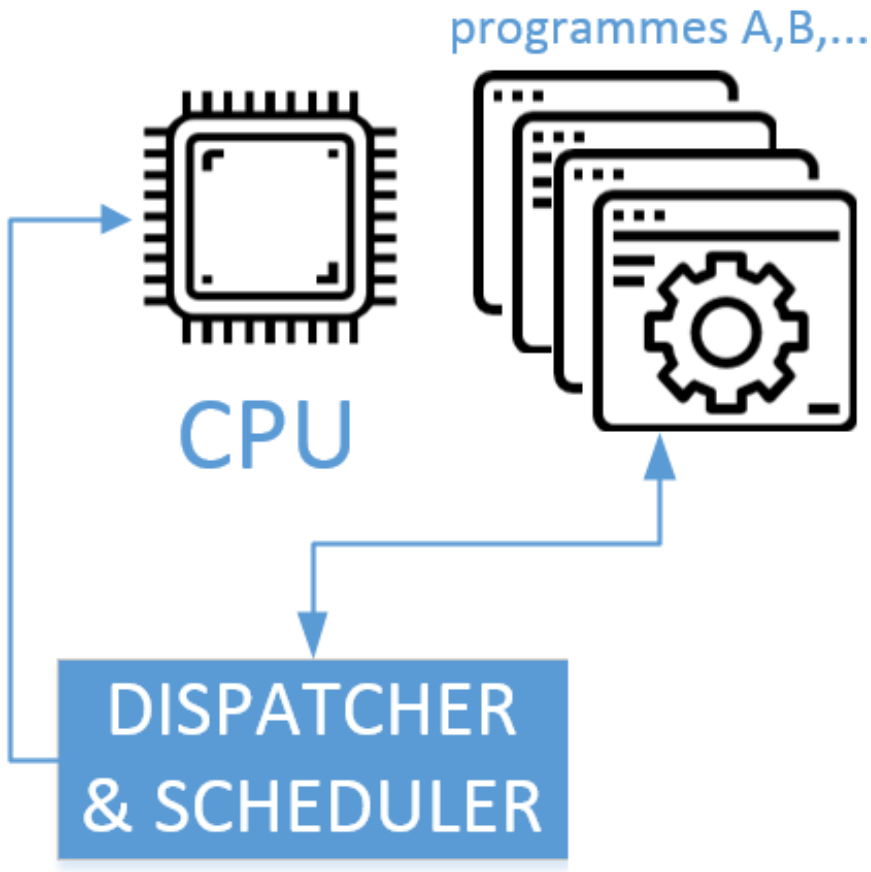
3)Une interruption est levée par le gestionnaire d'interruption et remonte au *dispatcher* qui stoppe son activité pour traiter l'interruption entrante.



4)Le logiciel de traitement de texte reçoit l'interruption → la lettre s'affiche à l'écran.

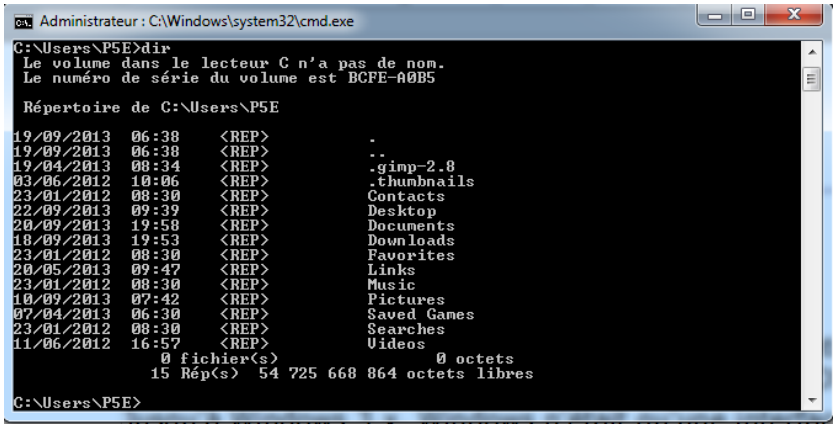


5)Le *dispatcher* et *scheduler* reprennent leur activité...

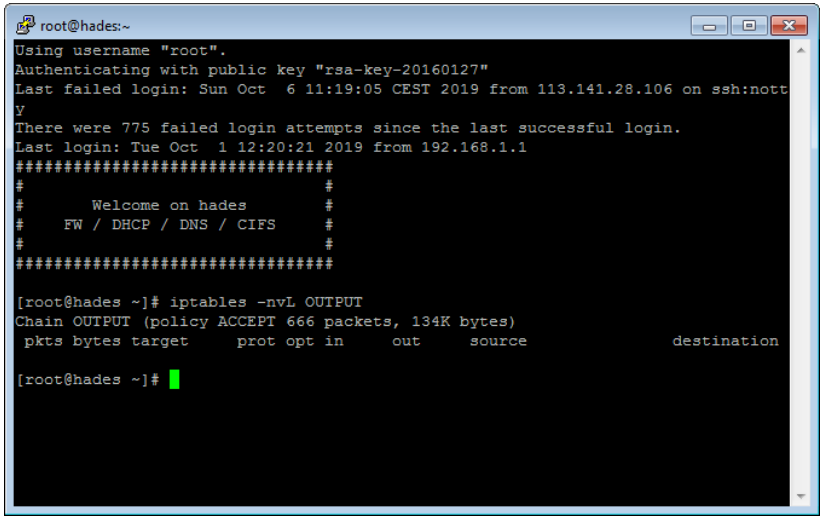


L'interpréteur de commande diffère selon le système d'exploitation utilisé.

- Sous Windows : l'invite DOS (Disk Operating System) a perduré jusqu'en 2009 où le PowerShell 1.0 a été introduit comme une mise à jour…



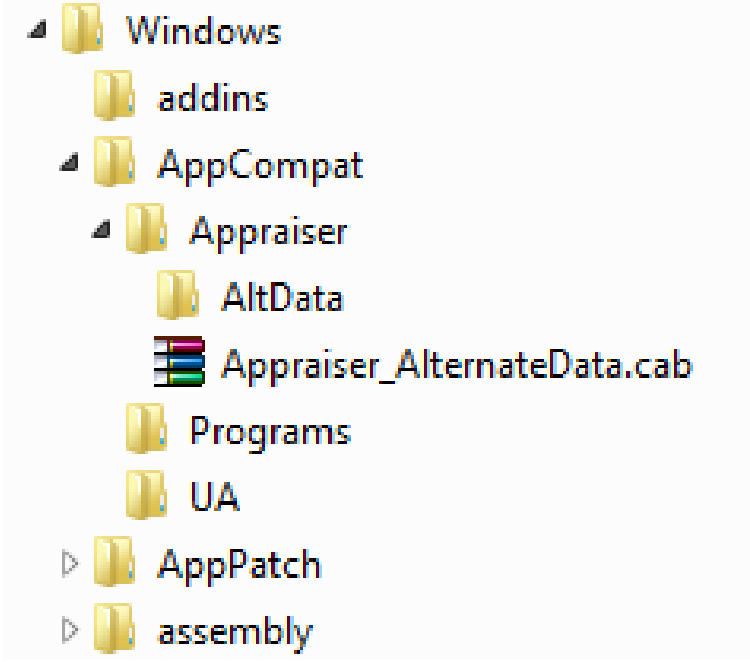
- Sous Linux : la ligne de commande est le moyen privilégié de communication avec le système. Le **B**ourne **A**gain **SH**ell (bash) est l'interpréteur le plus utilisé de nos jour.



Le système de fichiers (SGF) est vu comme une arborescence où les fichiers sont regroupés dans des répertoires.

Il y a un répertoire **racine** et des sous-répertoires.

Une telle organisation génère une hiérarchie de répertoires et de fichiers organisés en arbre.



Le SGF assure plusieurs fonctions :

- **Manipulation des fichiers** : création, lecture, modification et suppression des fichiers

- **Allocation d'espace** : le SGF alloue à chaque fichier un nombre variable de blocs mémoire (cluster).

- **Localisation des fichiers** : chaque fichier possède un ensemble d'informations descriptives (nom, adresse…), appelées métadonnées, qui permet de le retrouver.

- **Sécurité et contrôle des fichiers** : les fichiers sont protégés contre tout accès non autorisé ou mal intentionné.

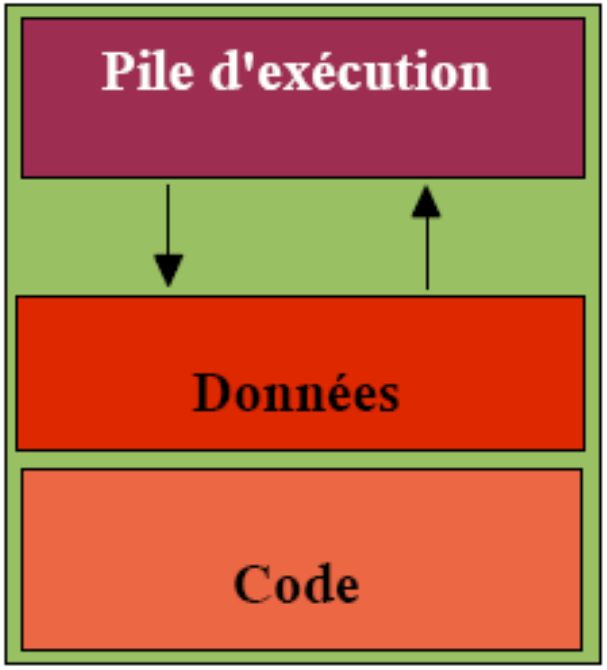
Les processus

Modélisation

Un processus modélise l'exécution d'un programme sur un processeur.

Un processus lourd dispose d'un espace mémoire qui lui est propre dans lequel il y a notamment :

- Les instructions à exécuter
- Les variables nécessaires à son exécution



Notion de contexte

Le contexte d'exécution d'un processus correspond aux données utilisées par ce processus.

Le contexte d'exécution est sauvegardé pour permettre l'interruption et la reprise du processus au même endroit (instruction) : c'est la commutation de contexte.

Processus lourd et léger

Un processus lourd peut être divisé en plusieurs sous-processus.

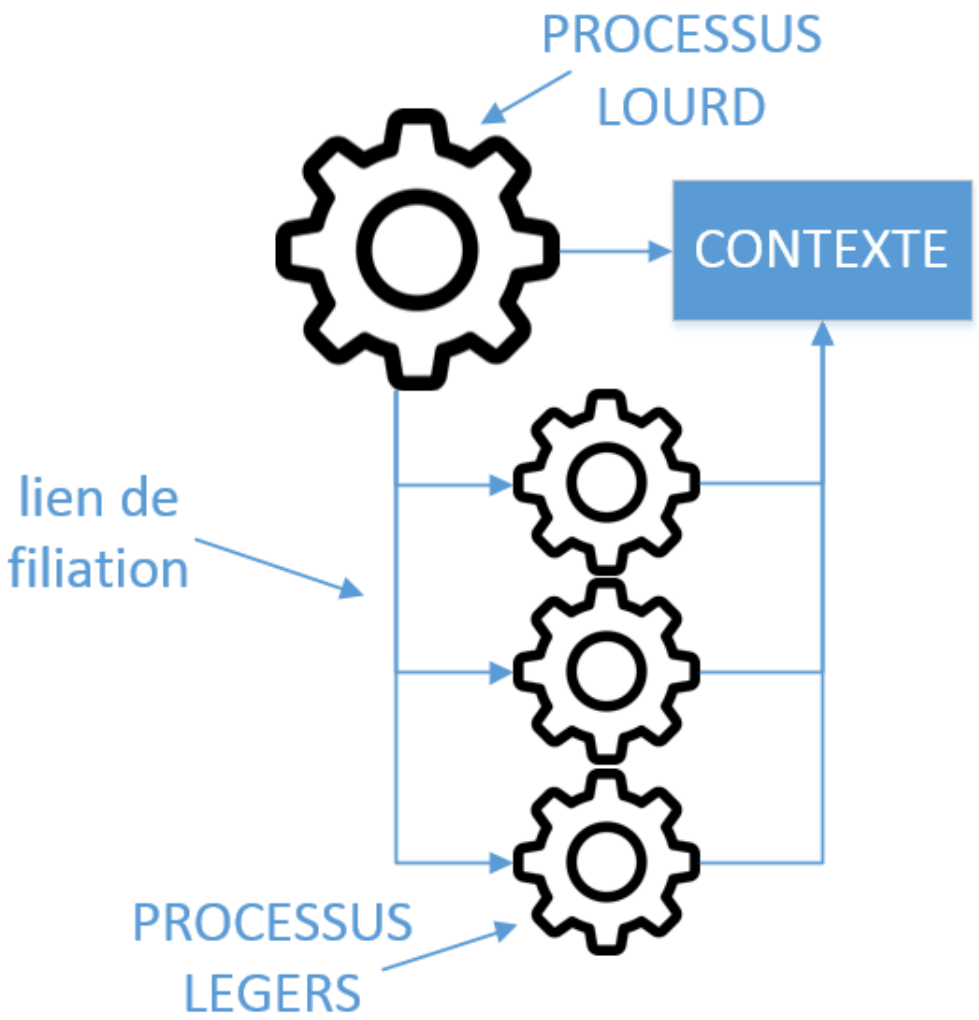
Les sous-processus possèdent un lien de filiation avec le processus créateur (processus père).

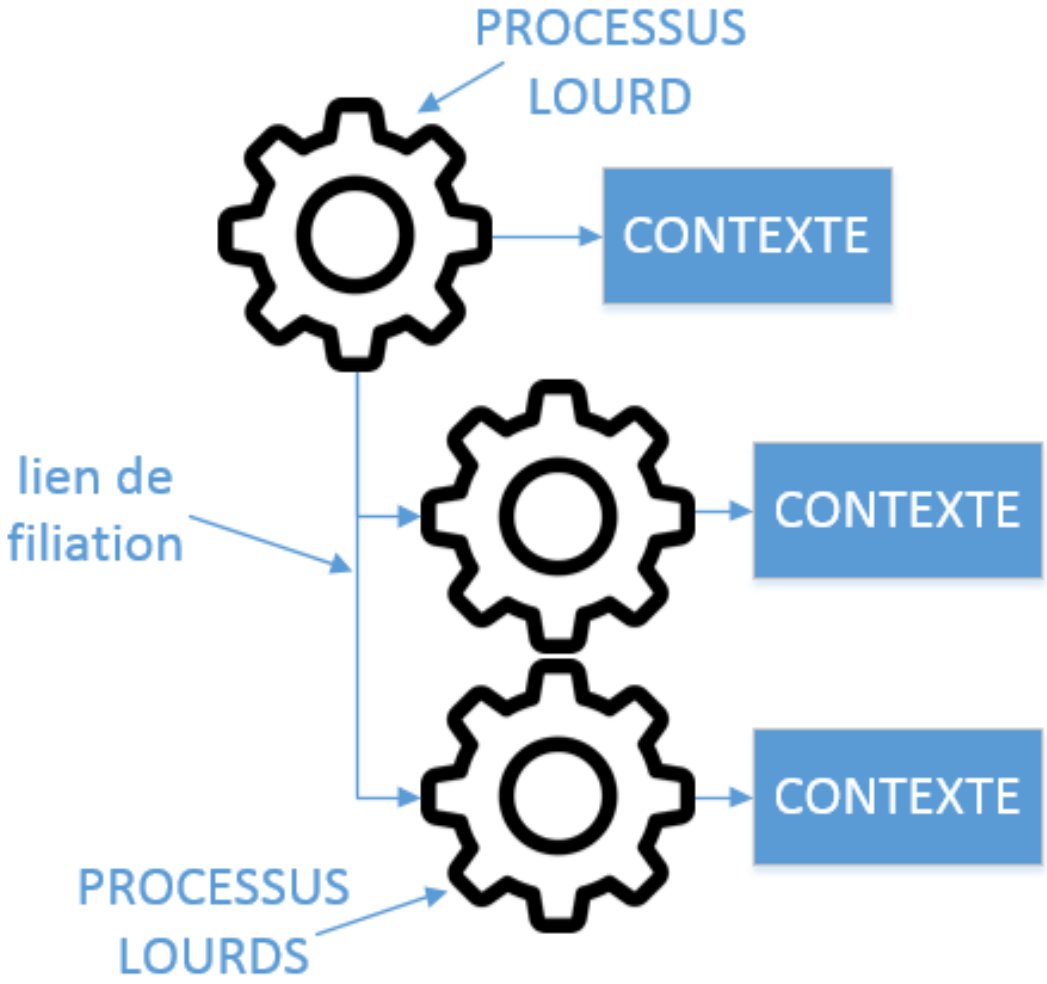
Les processus fils peuvent être légers (*thread*) ou lourds (*fork*).

Les *threads* partagent les ressources du processus lourd père.

Le contexte d'exécution des *forks* est une copie du contexte du père au moment de la création.

Processus lourd et léger

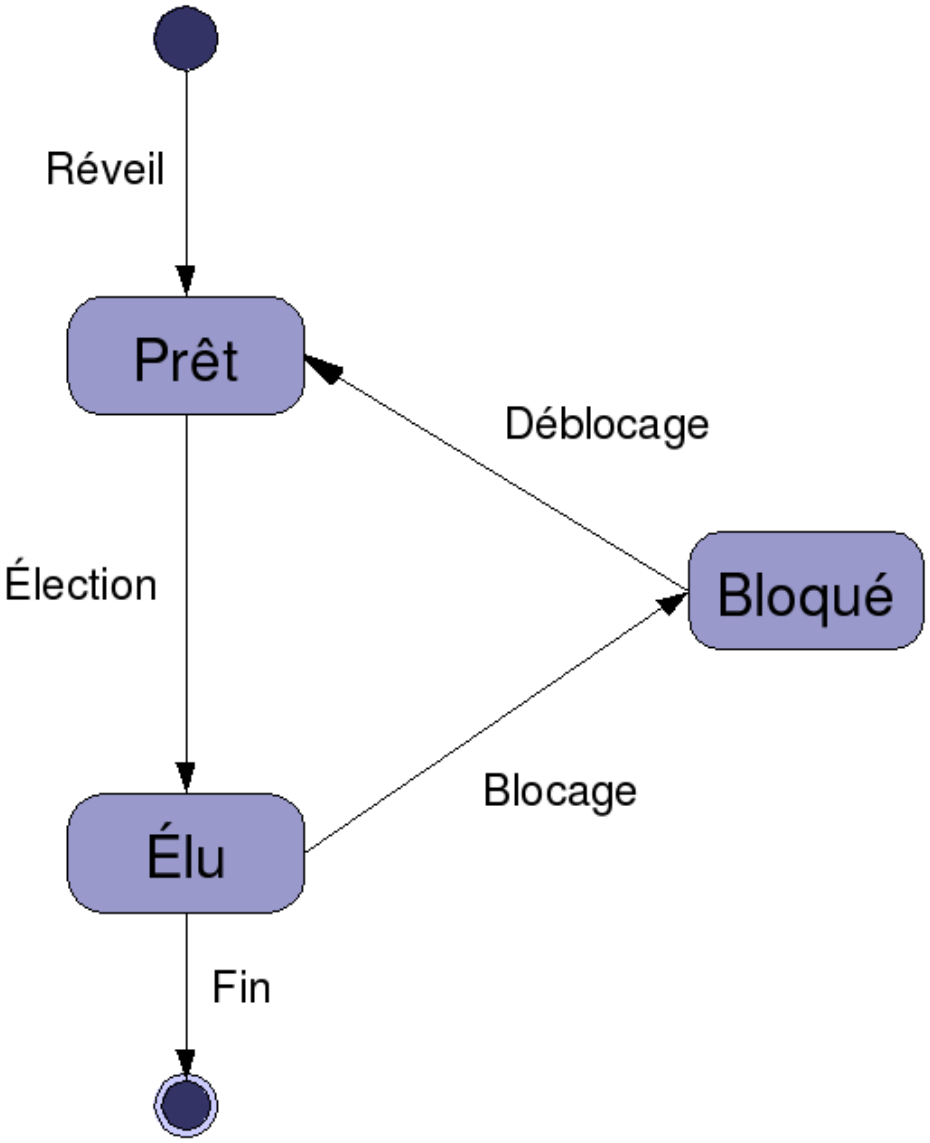




État des processus

De façon simplifiée, on peut imaginer les processus dans trois états :

1) Lorsque le processus est invoqué, il se **réveille** et attend son **élection**.
2) Une fois **élu** pour le prochain cycle d'horloge, le processus passe au processeur les instructions à exécuter.
3) Si l'instruction dure plusieurs cycles, le processus passe alternativement de l'état **bloqué** à l'état **élu**.
4) Lorsque toutes les instructions sont exécutées, le processus se **termine**.


Réveil
Prêt
Déblocage
Élection
Bloqué
Blocage
Élu
Fin

Ordonnancement des processus

Le duo *dispatcher / scheduler* définit l'ordre dans lequel les processus prêts utilisent les unités de calcul ainsi que la durée d'utilisation.

Ce choix est fait en utilisant un algorithme d'ordonnancement qui se doit de respecter au maximum les critères suivants :

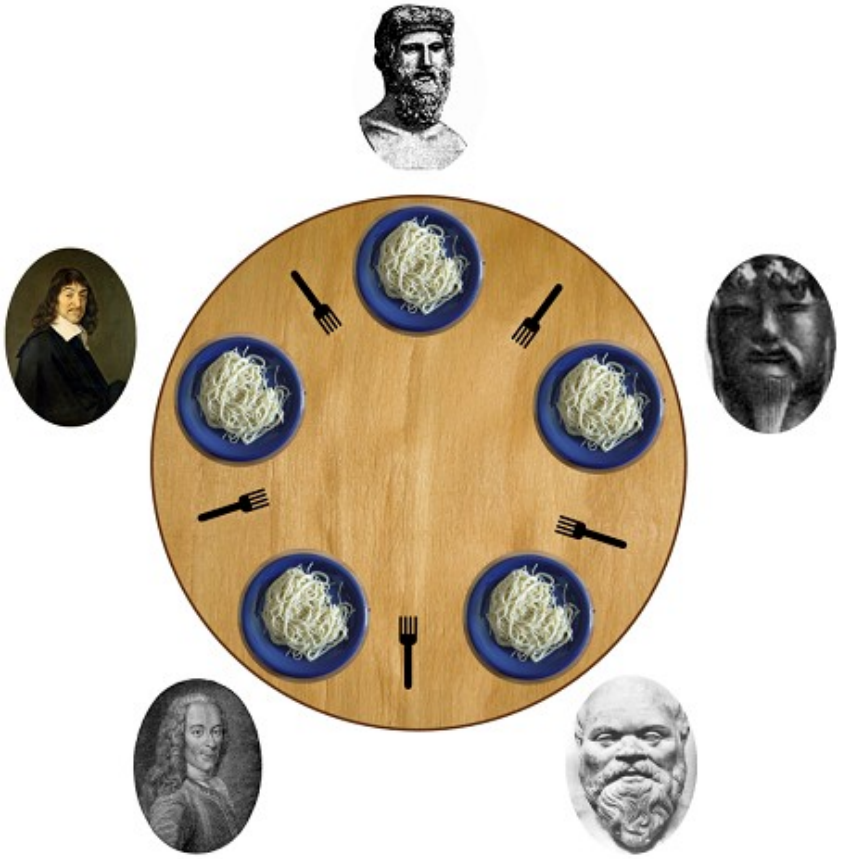
- **équité** : chaque processus reçoit sa part du temps processeur
- **efficacité** : le processeur doit travailler à 100 % du temps
- **temps de réponse** : à minimiser en mode interactif
- **temps d'exécution** : minimiser l'attente des travaux
- **rendement** : maximiser le nombre de travaux effectués par unité de temps

Ordonnancement des processus

Le problème du « dîner des philosophes » est un cas d'école classique sur le partage de ressources en informatique système.

Plusieurs philosophes (ici 5) se trouvent autour d'une table.

Chacun des philosophes a devant lui un plat de spaghettis et à gauche de chaque plat de spaghettis se trouve **une fourchette**.



Ordonnancement des processus

Un philosophe n'a que trois états possibles :

- **penser** pendant un temps indéterminé
- **être affamé** (pendant un temps déterminé et fini sinon il y a famine)
- **manger** pendant un temps déterminé et fini

Ordonnancement des processus

Des contraintes extérieures s'imposent à cette situation :

- **Quand un philosophe a faim**, il va se mettre dans l'état « **affamé** » et attendre que les **fourchettes** soient **libres.**

- Pour **manger**, un philosophe a besoin de **deux fourchettes** (à gauche et à droite de son assiette).

- Qi un philosophe n'arrive pas à s'emparer d'une fourchette, il reste affamé pendant un temps déterminé, en attendant de renouveler sa tentative.

Ordonnancement des processus : **dead lock**

Il suffit que chacun saisisse sa fourchette de gauche et, qu'ensuite, chacun attende que sa fourchette de droite se libère pour qu'aucun d'entre eux ne puisse manger, et ce pour l'éternité.

Cette situation s'appelle un *dead lock* ou interblocage.

Ordonnancement des processus : **temps partagé**

L'ordonnancement à temps partagé est présent sur la plupart des ordinateurs comme, par exemple, l'ordonnancement « **decay** » qui est celui par défaut sous ***Unix.***

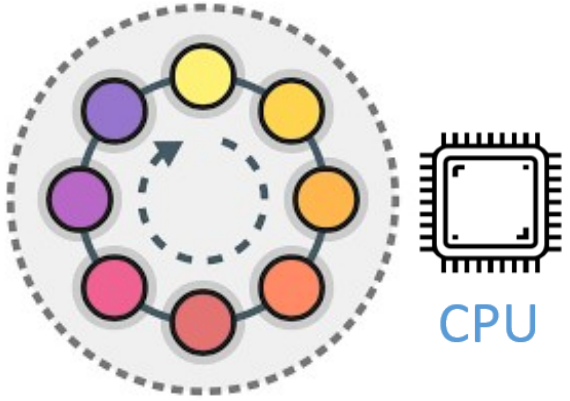
Il consiste en un système de **priorités** adaptatives qui privilégie les tâches interactives pour que leur temps de réponse soit bon.

Ordonnancement des processus : **round robin**

Le round-robin est similaire à un tourniquet : chaque processus est sur le tourniquet et ne fait que passer devant le processeur, à son tour et pendant un temps fini.

La non famine est garantie par l'utilisation d'un quantum de temps impossible à dépasser.

Un processus qui vient de finir d'utiliser le processeur (quantum écoulé) est placé en fin de liste



Communication inter processus

C'est l'**ensemble de mécanismes** permettant à des **processus concurrents** (ou distants) de **communiquer**.

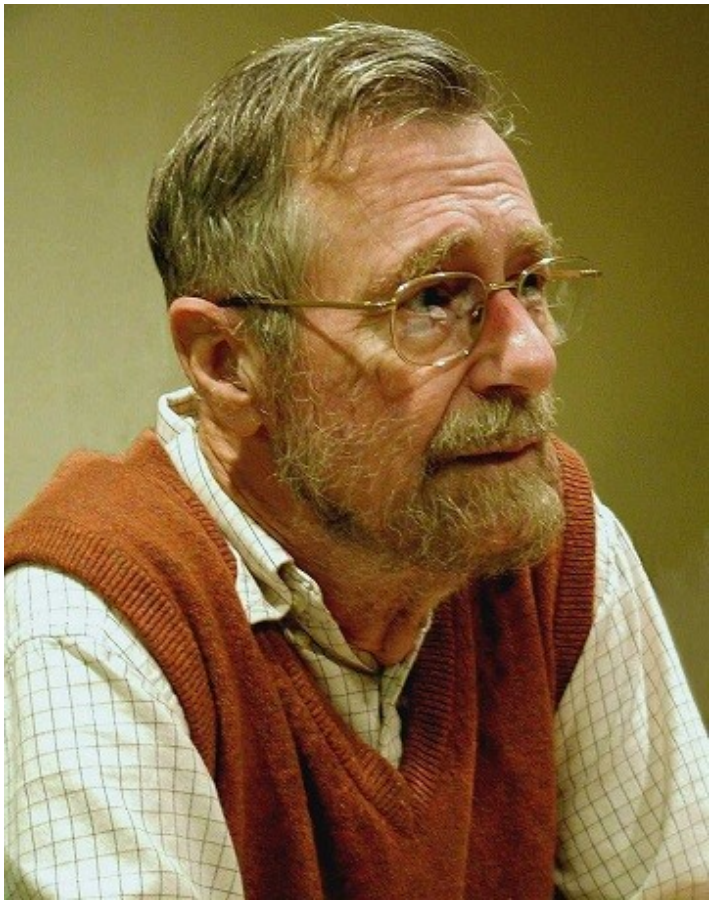
Ces mécanismes peuvent être classés en trois catégories :

- ceux permettant aux processus d'échanger des données
- ceux permettant de synchroniser les processus
- ceux offrant directement les deux

Mécanismes de synchronisation : le sémaphore

Un sémaphore est un mécanisme qui permet de restreindre l'accès à des ressources partagées dans un environnement de programmation concurrente.

Le sémaphore a été inventé par Edsger Dijkstra et utilisé pour la première fois dans le système d'exploitation ***THE Operating system***.

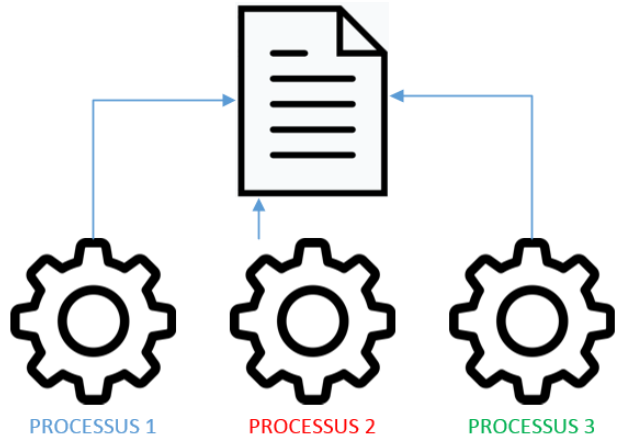


Prenons le cas suivant :

Plusieurs processus essaient d'écrire dans le même fichier.

Nous savons :

- que le dispatcher choisi les processus qui s'exécutent
- que l'on n'a aucun moyen sûr d'influencer le dispatcher



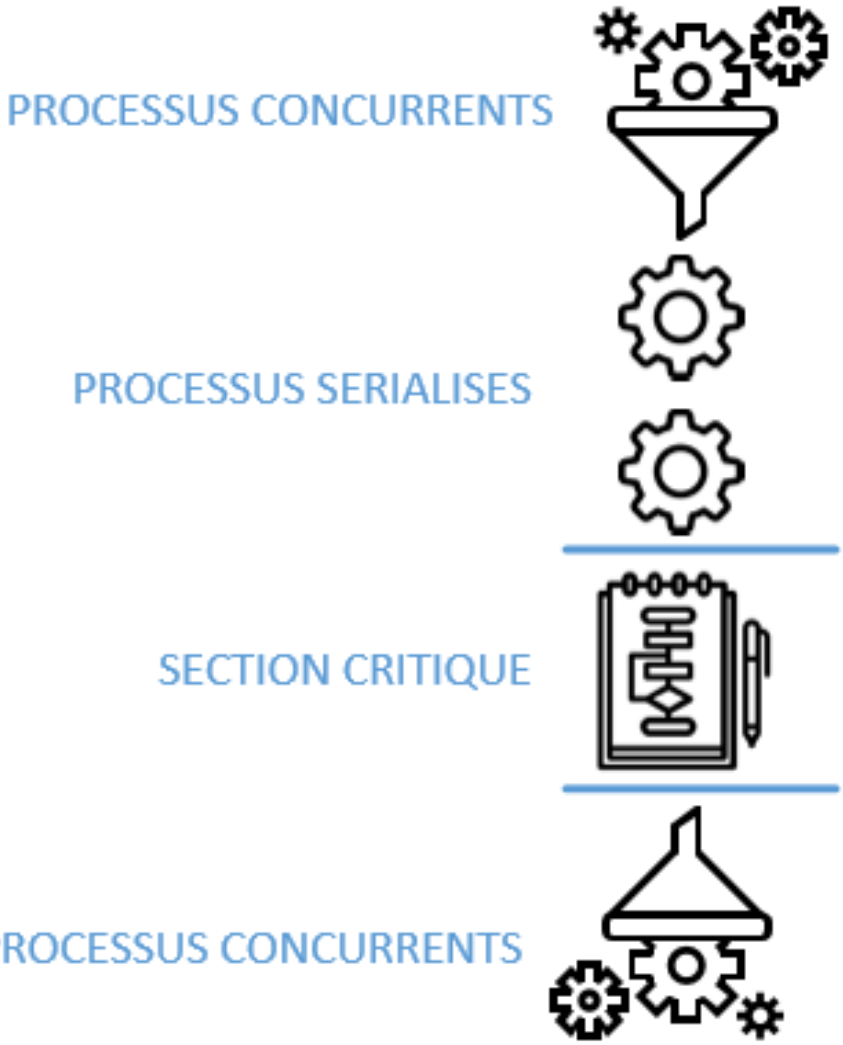
Le résultat sera un fichier texte qui contiendra toutes les lignes des processus mélangées, ces derniers s'exécutant aléatoirement en fonction du bon vouloir du *dispatcher.*

Le sémaphore va permettre de synchroniser les processus.

Il permet de créer un bloc de lignes de code insécable qui s'exécutent de manière atomique appelé **section critique.**

Dans la section critique on peut mettre autant de lignes de code que désiré.

Un processus ne peut entrer dans la section critique que si le processus précédent en est sorti.



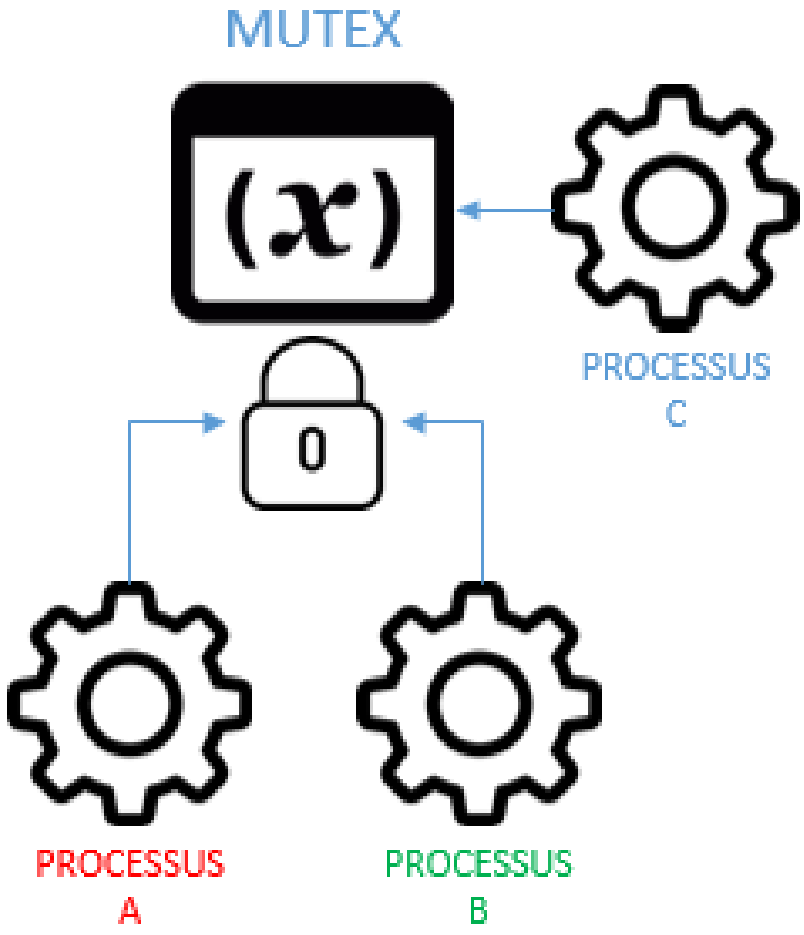
Mécanismes de synchronisation : l'exclusion mutuelle

Un *mutex* est un mécanisme qui permet à un processus de verrouiller l'accès à une variable.

Seul le processus qui a mis le verrou et autoriser à le retirer.

Les processus légers partagent le même contexte d'exécution et donc les mêmes variables.

Le *mutex* est un moyen de les synchroniser.

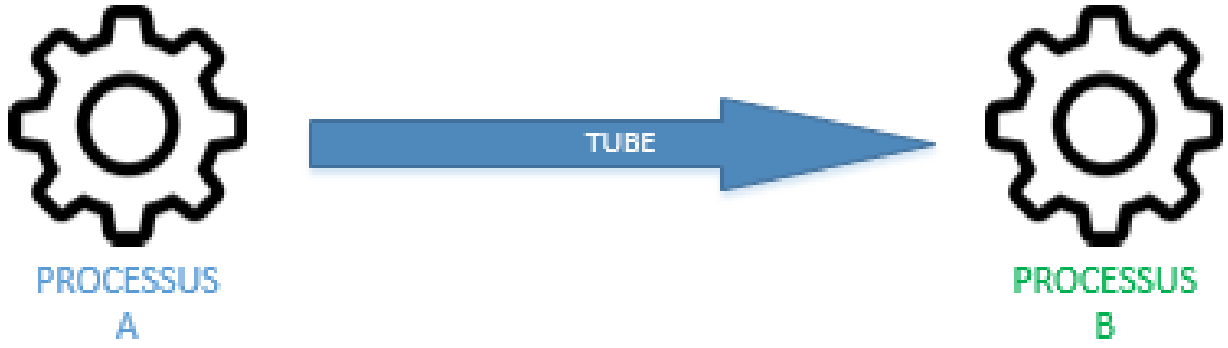


Mécanismes d'échange : les tubes ou pipes

Un tube est un canal par lequel les informations circulent de manière uni-directionnelle.

Un processus écrit dans l'entrée du tube et un autre processus lit les informations en sortie.

Cela fonctionne uniquement pour des processus qui sont sur la même machine.



Mécanismes d'échange : les sockets

Une socket permet de créer un canal de communication bidirectionnel.

Le processus créateur ou processus serveur utilise le réseau pour mettre sa socket en écoute de connexion.

Le processus client peut se connecter à la socket du processus serveur pour envoyer et recevoir des informations.